Das SSL-Apache Handbuch

DFN-PCA, Vogt-Kölln-Straße 30, D-22527 Hamburg, Universität Hamburg, FB Informatik - RZ

Version 1.07 07.03.2000

Apache ist ein HTTP-Server, der sich u.a. durch seine freie Verfügbarkeit für viele Betriebssysteme auszeichnet. Für den Apache existieren zwei Programme, die jeweils den Server um SSL-/TLS-Funktionalität ergänzen. Durch diese Ergänzung ist es möglich, verschlüsselte und authentisierte Verbindungen zwischen dem HTTP-Server und einem HTTP-Client zu realisieren. Beide Programme setzen ein betriebsbereites SSL-/TLS-Programmpaket *OpenSSL* voraus. Näheres zu Installation und Betrieb von OpenSSL findet sich in den *OpenSSL Handbüchern* `<http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/>` der DFN-PCA.

Inhaltsverzeichnis

1 HINWEIS **2**

2 Anmerkung **2**

3 Einleitung **2**

4 Installation der Software **3**

4.1 Das Paket `mm-1.0.12.tar.gz` 3

4.2 Das Paket `openssl-0.9.4.tar.gz` 5

4.3 Das Paket `mod_ssl-2.4.10-1.3.9` 7

4.4 Das Paket `apache_1.3.9` 7

4.4.1 Apache ohne DSO-Support 8

4.4.2 Apache mit DSO-Support 9

4.4.3 Installation des SSL-Apache durch Kopieren 10

4.4.4 Gruppen- und Benutzerrechte 11

4.5 Übersetzung von neuen Mod-SSL-Versionen 12

5 Betrieb des SSL-Apache **13**

5.1 Virtuelle Server 14

5.2 Erzeugen eines Server-Zertifikats 14

5.2.1 Erzeugen eines Zertifikats durch `make certificate` 14

5.2.2 Zertifizierung durch eine CA 16

5.3 Prüfung von Client-Zertifikaten 17

5.4 Widerrufslisten (CRLs) 18

5.5 Starten und Stoppen des Apache 20

A `http.conf` - Beispiel-Datei **20**

B Über dieses Dokument 34

C Links zu der dokumentierten Software 34

C.1 Browser-Relevantes 35

C.2 Weitere Tools 35

1 HINWEIS

Dieses Dokument (das „SSL-Apache Handbuch“) entstand in einem DFN-Projekt an der Universität Hamburg und wurde nach bestem Wissen und Gewissen verfaßt. Dennoch wird keine Haftung für die Korrektheit, Vollständigkeit oder Anwendbarkeit der hier beschriebenen Informationen und der vorgeschlagenen Maßnahmen übernommen. Ferner kann keine Haftung für eventuelle Schäden, entstanden durch die Anwendung der in diesem Dokument beschriebenen Anweisungen, übernommen werden. Die Verantwortung für die Verwendung der hier beschriebenen Verfahren und Programme liegt allein bei den die Installation durchführenden Personen.

2 Anmerkung

Diese Dokumentation ist für alle gedacht, die sich für einen SSL-fähigen Apache-Web-Server interessieren und genaueres über die Konfiguration des SSL-Teils eines solchen Servers wissen wollen. Diese Dokumentation kann und soll nicht die grundsätzliche Konfiguration eines „normalen“ Apache beschreiben. Daher werden Kenntnisse über Konfiguration und Betrieb eines solchen Servers vorausgesetzt.

Durch ein SSL-Paket (*Mod-SSL*) werden neue SSL-bezogene Server-Direktiven in den Apache gebracht. Diese SSL-Direktiven werden ausführlich in der Dokumentation des SSL-Pakets erläutert. Einige dieser Direktiven werden zusätzlich in den einzelnen Abschnitten des Kapitels 5 dieses Handbuchs dargestellt.

3 Einleitung

Eingangs einige Anmerkungen über den Zusammenhang zwischen *Apache-SSL* und *SSL-Apache*. Der Apache enthält keinerlei Kryptoroutinen. Wie im nächsten Abschnitt erläutert wird, ist es deshalb notwendig, Schnittstellen zu einer Krypto-Software (*OpenSSL*) in die Apache-Quellen zu „patchen“. Dazu stehen zwei Software-Pakete zur Verfügung: ein Paket, welches von *Ben Laurie* entwickelt wird, und ein anderes, das unter Koordination von *Ralf S. Engelschall* entwickelt wird. Der SSL-Server, der bei der Verwendung von Lauries Paket entsteht, wird üblicherweise **Apache-SSL** genannt. Der unter Verwendung von Engelschalls Paket erstellte Server heißt dagegen **Mod-SSL-Apache**, im Folgenden kurz *SSL-Apache*.

Beide bieten eine recht ähnliche Funktionalität. Das Vergleichen der Pakete in den entsprechenden Mailinglisten hat in der Vergangenheit zu recht „engagierten“ Auseinandersetzungen geführt. Daher soll hier nur kurz erläutert werden, warum für diese Dokumentation die Wahl auf das Paket von Engelschall fiel. Zum Zeitpunkt der Entscheidung unterstützte das Paket die Verwendung von Widerruflisten und bestach durch seine umfangreiche Dokumentation. Dies kann inzwischen durchaus auch der Fall für das Paket von Laurie sein. Die Empfehlung lautet daher, sich beide Pakete anzuschauen und dasjenige zu wählen, welches den eigenen Anforderungen am ehesten genüge tut.

4 Installation der Software

Bevor der SSL-Apache-Server in Betrieb genommen werden kann, müssen zunächst einige Software-Pakete übersetzt und installiert werden. Der SSL-Server liegt nicht als ein komplettes Quell-Paket vor. Das liegt daran, daß der Apache-HTTP-Server (ohne SSL-Funktionalität) von einer internationalen Gruppe entwickelt wird, darunter auch Mitglieder aus den USA. In den USA bestehen gewisse Export-Beschränkungen hinsichtlich Krypto-Software und -Schnittstellen. Daher erfolgt die Entwicklung des Servers ohne SSL-Funktionalität bzw. Schnittstellen. Die SSL-Funktionalität wird durch ein außerhalb der USA entwickeltes Software-Paket, *OpenSSL*, bereitgestellt. Die benötigten Schnittstellen zwischen Apache und dem OpenSSL-Paket werden schließlich durch das (ebenfalls außerhalb der USA entwickelte) *Mod-SSL*-Paket in die Apache-Server-Quellen „gepatched".

Die Übersetzung und Installation der einzelnen Pakete wird in den folgenden Abschnitten beschrieben.

Es sind im Prinzip vier Software-Pakete notwendig:

1. Optional, weil „lediglich" die Performanz des SSL-Apache-Servers verbessert wird, die Quellen zur Verwaltung von Shared-Memory, `mm-1.0.12.tar.gz`
2. Die Sourcen des TLS-/SSL-Pakets OpenSSL, `openssl-0.9.4.tar.gz`
3. Die Sourcen, die den Apache um die Schnittstellen zum TLS-/SSL-Paket erweitern, `mod_ssl-2.4.10-1.3.9.tar.gz`
4. Die Sourcen des HTTP-Servers Apache, `apache_1.3.9.tar.gz`

Informationen zu Installation und Betrieb des OpenSSL-Pakets finden sich wegen ihres Umfangs in einer eigenen *Dokumentation* `<http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/>`. Eine Kurzfassung zur Übersetzung und Installation von OpenSSL-0.9.4 findet sich weiter unten (4.2).

Alle in diesem Dokument gemachten Angaben zur Übersetzung und Konfiguration der einzelnen Software-Pakete beziehen sich auf das Betriebssystem SunOS 5.5.1 (Solaris 2.5.1) und den C-Compiler gcc-2.7.2.1.

4.1 Das Paket `mm-1.0.12.tar.gz`

Wie im vorigen Abschnitt angedeutet, ist dieses Paket für den Betrieb des Servers nicht unbedingt erforderlich, aber wegen der verbesserten Performanz sinnvoll.

Das MM-Paket (MM steht für „memory mapped") wurde von Ralf S. Engelschall entwickelt. Es stellt eine einheitliche Schnittstelle zur Verwaltung von Shared-Memory auf verschiedenen Plattformen zur Verfügung. Im Zusammenhang mit SSL-Apache kann bei Verwendung der MM-Bibliothek ein RAM-basierter SSL-Session-Cache anstelle eines Festplatten-Cache eingesetzt werden.

Auspacken der Quellen in einem Verzeichnis (z.B. `/usr/src`):

```
gzip -dc mm-1.0.12.tar.gz | tar -xvf -
```

Mit dem folgenden Befehl (nach dem Wechsel in das MM-Verzeichnis) wird das Paket konfiguriert:

- `cd mm-1.0.12`
- `sh ./configure --prefix=/usr/local`

Die Option `--prefix=`*xxx* ermöglicht die Angabe eines Pfades, unter dem nach der Übersetzung ein Konfigurationsskript (*xxx*`/bin`), die Bibliothek (*xxx*`/lib`), Header-Dateien (*xxx*`/include`) und Manual-Seiten (*xxx*`/man/man1` und *xxx*`/man/man3`) installiert werden. Soll später nicht das komplette Paket installiert werden sondern nur die Bibliothek, oder wenn die Installation durch Kopieren erfolgen soll, empfiehlt es sich, `--prefix=/usr/local` zu ersetzen durch `--prefix='pwd'`. Die spätere, in jedem Fall notwendige Installation durch `make install` erfolgt dann im aktuellen Verzeichnis, also dem Quell-Verzeichnis.

Durch den Aufruf von `make install` werden nicht nur Dateien installiert, sondern es wird auch die benötigte Bibliothek erzeugt. Daher ist der Aufruf nicht so ohne weiteres zu ersetzen. Die Angabe von `--prefix='pwd'` bietet die Möglichkeit einer lokalen Installation. Die benötigte Bibliothek kann dann in das gewünschte Verzeichnis kopiert werden. Entsprechendes gilt für die anderen Dateien.

Der Aufruf des `configure`-Befehls konfiguriert das Paket zur Erzeugung einer statischen und einer dynamischen Bibliothek. Soll die MM-Bibliothek statisch in den Apache gelinkt werden, muß die ausschließliche Erzeugung statischer Bibliotheken durch Konfiguration mit folgendem Kommando (anstelle des obenstehenden) veranlaßt werden:

```
sh ./configure --disable-shared --prefix=/usr/local
```

Die Übersetzung des Pakets erfolgt durch Aufruf von

```
make
```

Testen des Pakets durch Aufruf von

```
make test
```

Bei erfolgreichem Test wird nach einer Reihe von Test-Daten die folgende Meldung ausgegeben:

```
OK - ALL TESTS SUCCESSFULLY PASSED.
```

Abschließend kann das Paket mit folgenden Befehl installiert werden:

```
make install
```

Wurde bei obigem `Configure`-Kommando die Option `--prefix` nicht auf `'pwd'` gesetzt, ist die Installation abgeschlossen. Der restliche Teil dieses Abschnitts bezieht sich auf die Installation durch Kopieren der einzelnen Dateien.

Soll die statische MM-Bibliothek verwendet werden, ist eine Installation der Bibliothek im Prinzip nicht notwendig. Es reicht, bei der Konfiguration des Apache (s.u.) (4.4) die MM-Konfigurations-Variable `EAPI_MM` auf den Pfad zu dem `lib`-Verzeichnis mit der statischen Bibliothek zu setzen, z.B. `EAPI_MM=/usr/src/mm-1.0.12`. Unterhalb dieses Verzeichnisses findet sich auch das `include`-Verzeichnis mit der MM-Header-Datei. Soll die statische Bibliothek trotzdem installiert werden, sind die folgenden Kommandos notwendig:

- `cp lib/libmm.a lib/libmm.la /usr/local/lib`
- `chmod 644 /usr/local/lib/libmm.a /usr/local/lib/libmm.la`
- `cp include/mm.h /usr/local/include`
- `chmod 644 /usr/local/include/mm.h`

Soll der SSL-Apache die dynamische Bibliothek verwenden, muß die Bibliothek entweder in einem der üblichen Bibliotheks-Verzeichnisse installiert werden, oder es muß vor dem Start des SSL-Apache die Umgebungsvariable `LD_LIBRARY_PATH` um das Verzeichnis, welches die MM-Bibliothek enthält, erweitert werden. Soll die Installation der dynamischen Bibliothek z.B. in `/usr/local/lib` erfolgen, sind nachstehende Kommandos notwendig:

- `cp lib/libmm.so.10.0.12 /usr/local/lib`
- `cd /usr/local/lib && ln -s libmm.so.10.0.12 libmm.so`
- `ln -s libmm.so.10.0.12 libmm.so.10`
- `chmod 755 libmm.so.10.0.12`
- `cp include/mm.h /usr/local/include`
- `chmod 644 /usr/local/include/mm.h`

Damit ist die Installation des MM-Pakets abgeschlossen.

4.2 Das Paket `openssl-0.9.4.tar.gz`

Wie schon weiter oben erwähnt wurde, fällt dieser Abschnitt knapp aus, da es für dieses Paket eine eigene *Dokumentation* `<http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/>` gibt.

Auspacken der Quellen in einem Verzeichnis (z.B. `/usr/src`):

```
gzip -dc openssl-0.9.4.tar.gz | tar -xvf -
```

Mit dem folgenden Befehl (nach dem Wechsel in das OpenSSL-Quell-Verzeichnis) wird das Paket konfiguriert:

```
sh config --prefix=/usr/local
```

Die Option `--prefix=`*xxx* ermöglicht die Angabe eines Pfades, unter dem nach der Übersetzung Bibliotheken (*xxx*`/lib`), Header-Dateien (*xxx*`/include/openssl`), Binaries (*xxx*`/bin`), sowie verschiedene Dateien zur Verwaltung und Herausgabe von Zertifikaten (*xxx*`/ssl`) installiert werden. Genaueres dazu steht in der Dokumentation des Pakets.

Übersetzen des Pakets durch Aufruf von

```
make
```

Ein Test des übersetzten Pakets wird durch folgendes Kommando durchgeführt:

```
make test
```

Bei erfolgreichem Test wird am Ende des Testlaufs eine Meldung ähnlich der folgenden ausgegeben:

```
Signed certificate is in newcert.pem
newcert.pem: OK
OpenSSL 0.9.4 09 Aug 1999
built on: Thu Aug 12 12:14:31 MET DST 1999
platform: solaris-sparcv8-gcc
```

```
options:  bn(64,32) md2(int) rc4(ptr,char) des(idx,cisc,16,long) idea(int) blowfish(ptr)
compiler: gcc -DTHREADS -D_REENTRANT -mv8 -O3 -fomit-frame-pointer -Wall -DB_ENDIAN -DBN_DIV2W
'test' is up to date.
```

Abschließend kann das Paket mit dem folgenden Befehl installiert werden:

```
make install
```

Sollen die Dateien statt mit dem `install`-Befehl durch Kopieren installiert werden, sind folgende Kommandos notwendig:

(*$(SSLDIR)=Installationspfad von OpenSSL, z.B.* `/usr/local`):

- `mkdir $(SSLDIR)/include $(SSLDIR)/include/openssl $(SSLDIR)/bin $(SSLDIR)/lib`
- `cd /usr/src/openssl-0.9.4`
- `cp lib* $(SSLDIR)/lib/`
- `cp include/openssl/* $(SSLDIR)/include/openssl/`
- `cp apps/openssl tools/c_rehash $(SSLDIR)/bin/`

Optional (weil nicht zur Übersetzung des SSL-Apache erforderlich) kann jetzt noch eine Verzeichnis-Struktur zur Herausgabe von Zertifikaten mittels des OpenSSL-Pakets etabliert werden:

- `mkdir $(SSLDIR)/ssl $(SSLDIR)/ssl/private $(SSLDIR)/ssl/misc $(SSLDIR)/ssl/lib $(SSLDIR)/ssl/certs`
- `cd /usr/src/openssl-0.9.4/tools`
- `cp c_hash c_issuer c_info c_name .$(SSLDIR)/ssl/misc`
- `cd /usr/srs/openssl-0.9.4/apps`
- `cp CA.pl CA.sh der_chop $(SSLDIR)/openssl/misc`
- `cp apps/openssl.cnf $(SSLDIR)/ssl`

Die vernünftige Nutzung dieser Struktur, also letztlich der Betrieb einer CA, setzt Wissen über die Handhabung von Zertifikaten voraus. Das gilt besonders, wenn Zertifikate für den SSL-Apache und für Anwendungen selbst herausgegeben werden sollen. Aber auch, wenn „lediglich" ein Request für eine Zertifizierung durch eine externe CA erzeugt werden soll, ist es unumgänglich, sich etwas eingehender mit dem Paket zu beschäftigen.

Genaueres findet sich in einer anderen *Dokumentation* `<http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/>`.

Wie eine Serverzertifikat-Anforderung (*Request*) erzeugt werden kann, steht in einem Abschnitt weiter unten (5.2).

Damit ist die Installation des OpenSSL-Pakets abgeschlossen.

4.3 Das Paket `mod_ssl-2.4.10-1.3.9`

Das Mod-SSL-Paket enthält eine umfangreiche Dokumentation zur Installation und zum Betrieb des SSL-Apache. Sie ist im Quell-Verzeichnis des Pakets unter `pkg.ssldoc` zu finden.

In diesem Abschnitt wird die Konfiguration des Mod-SSL-Pakets beschrieben. Dieses Paket wird nicht separat übersetzt, da es dazu dient, die Apache-Quellen zu verändern bzw. zu erweitern. Es werden die notwendigen Schnittstellen zum OpenSSL-Paket in die Apache-Quellen „gepatched“. Die so veränderten Apache-Quellen müssen dann später übersetzt werden.

Auspacken der Quellen in einem Verzeichnis (z.B. in `/usr/src`):

```
gzip -dc mod_ssl-2.4.10-1.3.9.tar.gz | tar -xvf -
```

Bevor das Mod-SSL-Paket konfiguriert werden kann, müssen zunächst die Apache-Quellen entpackt werden (z.B. in `/usr/src`):

```
gzip -dc apache_1.3.9.tar.gz | tar -xvf -
```

Mit den folgenden Befehlen wird das Mod-SSL-Paket konfiguriert und gleichzeitig die notwendigen Veränderungen am Apache-Quellcode vorgenommen:

- `cd mod_ssl-2.4.10-1.3.9`
- `sh ./configure --with-apache=../apache_1.3.9`

Bei der Option `--with-apache=` muß der Pfad angegeben werden, der auf die zuvor entpackten Apache-Quellen weist.

Verlief die Ausführung des vorstehenden Kommandos erfolgreich, wird eine abschließende Meldung ausgegeben:

```
Done: source extension and patches successfully applied.
```

4.4 Das Paket `apache_1.3.9`

Das Apache-Paket enthält eine umfangreiche Dokumentation des Servers im HTML-Format. Die Dokumentation befindet sich im Quell-Verzeichnis unterhalb des Verzeichnisses `htdocs`. Ebenfalls in diesem Verzeichnis befindet sich nach dem „Patchen“ der Apache-Quellen die Mod-SSL-Dokumentation.

Der Apache-Web-Server besteht aus verschiedenen Modulen, die die Funktionalität des Servers bestimmen. Die Auswahl dieser Module ist abhängig vom vorgesehenen Einsatz des Servers: z.B. davon, ob CGI-Skripte oder Benutzerverzeichnisse zugelassen werden sollen. Zur Verwendung von CGI-Skripten ist dann das Modul `mod_cgi` erforderlich, und zur Unterstützung von User-Verzeichnissen das Modul `mod_userdir`. Daher ist es unumgänglich, sich mit den Modulen und deren Funktionalität auseinanderzusetzen. Je nach gewünschtem Einsatz des Servers werden die Module bei dem untenstehenden Konfigurationsbefehl mit angegeben.

Die im Folgenden vorgestellte Konfiguration erlaubt u.a. den Betrieb eines SSL-fähigen HTTP-Servers und die Verwendung von CGI-Skripten. Die vorgestellte Konfiguration unterstützt aber beispielsweise nicht die automatische Generierung von Index-Dateien in FTP-Verzeichnissen. Benutzerverzeichnisse werden ebenfalls nicht unterstützt.

Zur Übersetzung kann der Apache auf zwei verschiedene Arten konfiguriert werden: mit Unterstützung für zur Laufzeit ladbare Module, den sogenannten „DSO-Support“ (DSO, *Dynamic Shared Object*), und ohne diesen DSO-Support.

In Hinsicht auf die SSL-Fähigkeit des Servers unterscheiden sich die beiden Konfigurationsvarianten in folgender Weise:

- *Werden DSO-Module vom Server nicht unterstützt*, müssen die Original-Server-Quellen bei Herausgabe einer neuen Mod-SSL-Version erneut „gepatched“ und übersetzt werden.
- *Unterstützt der Server DSO-Module*, kann die Laufzeitkonfiguration des Servers (festgelegt in der Datei `httpd.conf`) einfach geändert werden. Erfordert eine neue Konfigurationsdirektive die Funktionalität eines nicht geladenen Moduls, kann dieses nachgeladen werden. Nicht benötigte Module werden nicht geladen. Voraussetzung ist, daß sämtliche Module übersetzt vorliegen.

  Neue Mod-SSL-Version können übersetzt und anstelle des alten SSL-Moduls geladen werden. So muß nur das Modul und nicht der ganze Server übersetzt werden. Genaueres steht im Abschnitt weiter unten (4.5).

  Nachteilig an der DSO-Variante sind leichte Performanz-Einbußen zur Laufzeit und eine etwas längere Startdauer des Servers.

Denkbar ist auch, zunächst einen DSO-Server mit sämtlichen Modulen zu übersetzen. Mit diesen können dann verschiedene Modul-Konfigurationen getestet werden. Der letztlich eingesetzte Server könnte dann nur mit den benötigten Modulen in der Nicht-DSO-Variante übersetzt und anschließend eingesetzt werden.

Es sei hier angemerkt, daß unnötige Module Speicherplatz kosten (in der DSO-Variante nur, wenn sie geladen werden) und das Risiko erhöhen, daß der Server durch falsche Konfiguration kompromittierbar wird.

4.4.1 Apache ohne DSO-Support

Die gewünschten Module werden beim Aufruf des Konfigurations-Befehls `configure` angegeben. Wegen der verschiedenen Möglichkeiten, diese Module zusammenzustellen, wird im Folgenden eine Beispielkonfiguration aufgeführt, die mit der Server-Konfigurationsdatei im Anhang (A) funktioniert. Die unten angegebenen Module ermöglichen den Betrieb eines SSL-Servers und erlauben die Verwendung von CGI-Skripten.

Die Quellen des Apache liegen, wie im obigen Abschnitt (4.3) beschrieben, „gepatched“ vor, z.B. unter `/usr/src/apache_1.3.9`. Nach Wechsel in das Apache-Quell-Verzeichnis erfolgt die Konfiguration durch das folgende, etwas längere Kommando:

```
env SSL_BASE=/usr/local \
EAPI_MM=/usr/local \
CC="gcc" OPTIM="-O3 -mv8 -fomit-frame-pointer" \
sh ./configure --prefix=/usr/local/apache \
             --disable-module=all        --disable-rule=SSL_COMPAT \
             --enable-module=env \
             --enable-module=log_config  --enable-module=mime \
             --enable-module=negotiation --enable-module=dir \
             --enable-module=cgi         --enable-module=actions \
             --enable-module=access      --enable-module=setenvif \
             --enable-module=alias       --enable-module=ssl
```

Die Variable `SSL_BASE` weist auf den Zweig des Dateisystems, in dem das OpenSSL-Paket installiert bzw. übersetzt wurde. Das gleiche gilt für `EAPI_MM` in Bezug auf die MM-Bibliothek. Der bei der Option `--prefix=` angegebene Pfad veranlaßt, daß bei einem späteren Aufruf von `make install` sämtliche Apache-Dateien unter diesem Verzeichnis in verschiedene Unterverzeichnisse installiert werden.

Die einfachste Möglichkeit, ein feiner angepaßtes Installations-Layout zu bekommen, besteht in der Anpassung der Datei `config.layout` im Apache-Quellverzeichnis (`apache_1.3.9`). Zu dieser Layout-Datei kann ein eigenes benanntes Layout hinzugefügt werden, welches statt mit der Option `--prefix=`*xxx* mit der Option `--with-layout=`*layoutname* angegeben wird. Es ist auch möglich, die einzelnen Dateien nach der Übersetzung durch Kopieren an die gewünschte Stelle zu kopieren. Diese Variante wird im untenstehenden Abschnitt (4.4.3) vorgestellt.

Die Übersetzung wird durch folgendes Kommando gestartet:

```
make
```

Die Übersetzung sollte ohne Fehler oder Warnungen erfolgen.

4.4.2 Apache mit DSO-Support

Die in diesem Abschnitt vorgestellte Konfiguration konfiguriert die Server-Quellen so, daß sämtliche möglichen Module des Servers übersetzt und installiert werden. Welche Module dann tatsächlich für den Betrieb des Servers geladen werden, wird über die Konfigurationsdatei (siehe Anhang (A)) festgelegt. (Alternativ können auch nur die tatsächlich benötigten Module durch mehrfache Verwendung der Option `--enable-shared=`*Modulname* bei untenstehendem Konfigurations-Kommando angegeben werden.)

Damit der Apache die Bibliotheken des MM-Pakets findet, muß die Umgebungsvariable `LD_LIBRARY_PATH` den Pfad `/usr/local/lib` enthalten.

Der folgende Befehl konfiguriert die Apache-Quellen so, daß sämtliche Apache-Module und das SSL-Modul kompiliert werden:

```
env SSL_BASE=/usr/local \
    EAPI_MM=/usr/local \
    CC='gcc' OPTIM='-O3 -mv8 -fomit-frame-pointer' \
sh ./configure \
   --prefix=/usr/local/apache \
   --enable-shared=ssl \
   --disable-rule=SSL_COMPAT \
   --enable-shared=max \
   --enable-shared=remain
```

Sollen einzelne Module von der Konfiguration ausgenommen werden, kann die Konfigurations-Option `--disable=`*Modulname* eingesetzt werden:

```
env SSL_BASE=/usr/local \
    EAPI_MM=/usr/local \
    CC='gcc' \
    OPTIM='-O3 -mv8 -fomit-frame-pointer' \
sh ./configure \
   --prefix=/usr/local/apache \
   --enable-shared=ssl \
   --enable-shared=max \
   --enable-shared=remain \
   --disable-shared=auth_db \
   --disable-module=auth_db \
```

```
--disable-shared=auth_digest \
--disable-module=auth_digest
```

Die Variable `SSL_BASE` weist auf den Zweig des Dateisystems, in dem das OpenSSL-Paket installiert bzw. übersetzt wurde. Das gleiche gilt für `EAPI_MM` in Bezug auf die MM-Bibliothek. Der bei der Option `--prefix=` angegebene Pfad veranlaßt, daß bei einem späteren Aufruf von `make install` sämtliche Apache-Dateien unter diesem Verzeichnis in verschiedene Unterverzeichnisse installiert werden.

Die einfachste Möglichkeit, ein feiner angepaßtes Installations-Layout zu bekommen, besteht in der Anpassung der Datei `config.layout` im Apache-Quellverzeichnis (`apache_1.3.9`). Zu dieser Layout-Datei kann ein eigenes benanntes Layout hinzugefügt werden, welches statt der Option `--prefix=`*xxx* mit der Option `--with-layout=`*layoutname* angegeben wird. Es ist auch möglich, die einzelnen Dateien nach der Übersetzung an die gewünschte Stelle zu kopieren. Diese Variante wird in dem nachfolgenden Abschnitt (4.4.3) vorgestellt.

Die Übersetzung erfolgt durch Aufruf von

```
make
```

Es ist möglich, die gewünschten Module alle explizit aufzuführen. Die im vorigen Abschnitt (4.4.1) vorgestellte Nicht-DSO-Konfiguration wird zur DSO-Konfiguration, wenn in der Konfiguration alle `--enable-module` Optionen durch `--enable-shared` ersetzt werden.

4.4.3 Installation des SSL-Apache durch Kopieren

Nachfolgend wird die Installation eines „normalen“ HTTP-Servers auf Port 80 und eines SSL-Servers auf Port 443 beschrieben.

Die vorgestellte Installation ist beispielhaft zu verstehen und kann von Fall zu Fall sehr unterschiedlich aussehen. Die Installation kann durch `make install` automatisch erfolgen. Dabei werden aber auch Dateien installiert, die eher für einen Test-Betrieb gedacht sind. Daher wird auf diese Option hier nicht weiter eingegangen und im folgenden gezeigt, wie die notwendigen Dateien kopiert werden.

Die Installation der Dateien und die Betriebs-Konfiguration des Servers sind voneinander abhängig. Die Betriebs-Konfiguration des Servers erfolgt über eine Konfigurationsdatei. Die Beispiel-Konfigurationsdatei im Anhang (A) beschreibt den Betrieb eines Nicht-SSL-Servers (üblicherweise auf Port 80) und eines SSL-Servers auf Port 443. Dafür werden zwei Hauptverzeichnisse angelegt. Das erste nimmt innerhalb von fünf oder sechs Unterverzeichnissen die Konfigurationsdateien, Log-Dateien, CGI-Skripte, Icons, SSL-Dateien (Zertifikate u.a.) und eventuell die Apache-Module (eines DSO-Servers) auf. Das andere Hauptverzeichnis enthält zwei Unterverzeichnisse, die für je einen Server das sogenannte „Document-Root-Verzeichnis“ bilden. Diese Unterverzeichnisse nehmen die HTML-Seiten auf, die von dem jeweiligen Server gesendet werden sollen.

Zum Aufbau der Verzeichnisstruktur sind folgende Befehle notwendig:

(`$(APADIR)`=*Apache-Installationspfad*, z.B. `$(APADIR)=/usr/local/apache`),
(`$(WWW)`=*HTML-Root-Verzeichnis*, z.B. `$(WWW)=/var/www`)

- `mkdir $(APADIR) $(APADIR)/conf $(APADIR)/logs $(APADIR)/icons $(APADIR)/cgi-bin $(APADIR)/ssl $(APADIR)/ssl/cacerts $(APADIR)/ssl/crls`
- `mkdir $(WWW) $(WWW)/www80 $(WWW)/www443`

Wurde der Server mit DSO-Unterstützung übersetzt, muß zusätzlich noch ein Verzeichnis `libexec` angelegt werden, welches die übersetzten Apache-Module aufnimmt.

- mkdir $(APADIR)/libexec

Die man-Seiten, die ausführbaren Dateien bzw. Skripte sowie die HTML-Dokumentation des Servers werden im entsprechenden Zweig des Dateibaums installiert, also z.B. unterhalb /usr/local/man bzw. in /usr/local/bin und /usr/local/doc. Die Dokumentation bzw. die Manual-Seiten sind zur Laufzeit nicht erforderlich und können bei Bedarf weggelassen werden.

Jetzt können die Dateien kopiert werden.

Für einen ersten Test des Servers kann die mit dem Apache und die mit dem Mod-SSL-Paket gelieferte Hauptseite kopiert werden.

1. cd /usr/src/apache_1.3.9
2. cp src/httpd /usr/local/bin
3. cd src/support
4. cp ap apachectl apxs dbmmanage htdigest htpasswd log_server_status logresolve split_logfile suexec /usr/local/bin
   Diese Dateien müssen nicht alle vorhanden sein, da die Übersetzung in Abhängigkeit von den gewählten Modulen erfolgt.
5. cd /usr/src/apache_1.3.9
6. cp conf/mime.types $(APADIR)/conf
7. cp conf/httpd.conf-dist $(APADIR)/conf/httpd.conf
8. cp support/*.1 /usr/local/man/man1
9. cp support/*.8 /usr/local/man/man8
10. cp htdocs/index.html $(WWW)/www80
11. cp htdocs/manual/mod/mod_ssl/index.html $(WWW)/www443
12. cp htdocs/apache_pb.gif $(APADIR)/icons

Wurde der Server mit DSO-Unterstützung übersetzt, müssen noch die Module kopiert werden. Da diese zum Teil auf diverse Unterverzeichnisse verteilt sind, wird das find-Kommando eingesetzt:

- cp ‘find . -name "*.soprint‘ $(APADIR)/libexec/

Wichtig:
In jedem Fall ist vor Aufnahme des regulären Betriebs des Servers die Konfigurationsdatei httpd.conf im Verzeichnis $(APADIR)/conf anzupassen. Eine Beispieldatei findet sich im Anhang (A).

4.4.4 Gruppen- und Benutzerrechte

Aus Sicherheitsgründen sollte für den Server unbedingt ein eigener Benutzer (hier httpsd) und eine eigene Gruppe (hier www) eingerichtet werden. Der Server muß vom Benutzer root gestartet werden, damit der Server sich an die privilegierten Ports (80 und 443) binden kann. Anschließend werden Kind-Prozesse gestartet, die die Anfragen bedienen. Der Eltern-Prozeß läuft unter der User-ID root, die Kind-Prozesse unter der in der Konfigurationsdatei angegebenen (weniger privilegierten) User- bzw. Group-ID, also z.B. httpsd bzw.

`www`. Weiterhin ist es sinnvoll, einen eigenen User (hier `wwwadm`) zur Pflege der Web-Seiten einzurichten. Der User `wwwadm` gehört dann derselben Gruppe wie der Server an (hier `www`).

Im folgenden ein Vorschlag zum Setzen von Benutzern, Gruppen und den Rechten.

Es soll hier nochmal betont werden, daß große Aufmerksamkeit bei Vergabe der Rechte und Festlegung der Gruppen walten sollte. Ein falsch konfigurierter Server eröffnet möglicherweise einen nicht autorisierten Zugang zum Rechner! Insbesondere sollten Dateien, die von Prozessen mit der UID root gelesen werden, nicht von anderen geschrieben werden können.

Setzen von Benutzer und Gruppe:

- `chown -R root:root $(APADIR)`
- `chown -R wwwadm:www $(APADIR)/icons $(APADIR)/cgi $(APADIR)/ssl $(www)/www80 $(WWW)/www443`

Setzen der Rechte:

- `chmod -R 755 $(APADIR)`
- `chmod -R 700 $(APADIR)/logs $(APADIR)/ssl`
- `chmod 750 $(APADIR)/conf $(APADIR)/icons $(APADIR)/cgi $(APADIR)/crls \`
  `$(APADIR)/cacerts $(WWW)/www80 $(WWW)/www443`
- `chmod 600 $(APADIR)/conf/*`
- `chmod 640 $(APADIR)/icons/* $(WWW)/www80/* $(WWW)/www443/*`

Für das DSO-Module-Verzeichnis:

- `chown root:root $(APADIR)/libexec/*`
- `chmod 755 $(APADIR)/libexec/*`

Zum Starten bzw. Stoppen des Servers siehe untenstehenden Abschnitt (5.5).

4.5 Übersetzung von neuen Mod-SSL-Versionen

Bevor ein neues Mod-SSL-Paket mit dem Apache in Betrieb genommen wird, sollte auf jeden Fall die Datei `CHANGES` *des Mod-SSL-Pakets gelesen werden.* Möglicherweise wurden nicht nur Fehler beseitigt, sondern auch neue Konfigurationsdirektiven eingeführt oder alte Direktiven geändert. Entsprechend müßte dann vor dem Start des Servers eine Anpassung der Konfigurationsdatei vorgenommen werden.

Wurde der Apache *ohne DSO-Support* installiert, also die Module statisch in den Apache eingebunden, muß der Apache zusammen mit dem neuen Mod-SSL-Paket komplett übersetzt und installiert werden (wie oben beschrieben (4.4)). Dazu müssen die Apache-Quellen erneut ausgepackt werden, und nicht etwa die „alten“, schon „gepatchten“ Quellen wiederverwendet werden.

Wurde der Apache *mit DSO-Support und SSL-Modul* installiert, vereinfacht sich die Installation der neuen Mod-SSL-Version. Das Vorgehen wird im folgenden beschrieben.

Es wird angenommen, daß die Installation des OpenSSL-Pakets und des Apache unter `/usr/local` erfolgte. Insbesondere das mitgelieferte Perl-Skript mit Namen `apxs` muß unter dem angegebenen Pfad zur Verfügung stehen. In diesem Skript sind u.a. die Pfade eingetragen, in denen der Apache bzw. seine Komponenten installiert wurden.

- `cd /usr/src`
- `gzip -dc mod_ssl-2.4.`x`-1.3.9.tar.gz`
- `cd mod_ssl-2.4.`x`-1.3.9`
- ```
  env CC=gcc sh ./configure \
                --with-apxs=/usr/local/bin/apxs \
                --with-ssl=/usr/local
  ```
- `make`

Vor Ausführung des nächsten Befehls sollte das alte Modul umbenannt werden, um im Zweifelsfall die letzte (lauffähige) SSL-Apache-Version wiederherstellen zu können. Anschließend wird dann das neue Modul kopiert:

- `cp /usr/local/apache/libexec/libssl.so /usr/local/apache/libexec/libssl.so.old`
- `cp mod_ssl-2.4.`n`-1.3.9/pkg.sslmod/libssl.so /usr/local/apache/libexec/`
- `chown root:root $(APADIR)/libexec/libssl.so`
- `chmod 755 /usr/local/apche/libexec/libssl.so`

Dann muß der Server gestoppt und gestartet werden:

1. `kill -TERM ‘cat /usr/local/apache/logs/httpsd.pid‘`
2. `/usr/local/bin/httpd -DSSL`

Genaueres zum Starten bzw. Stoppen steht in einem Abschnitt weiter unten (5.5).

5 Betrieb des SSL-Apache

Bevor der SSL-Apache in Betrieb genommen werden kann, muß normalerweise ein Schlüsselpaar nebst Zertifikat für den Server erzeugt werden. Ohne dieses Server-Zertifikat kommt keine SSL- und somit keine verschlüsselte Verbindung zustande. Das Zertifikat enthält den Public-Key sowie u.a. den Namen des Servers. Bei einem SSL-Verbindungswunsch des Client wird das Server-Zertifikat vom Server an den Client geschickt. Wurde das Zertifikat durch eine dem Client bekannte CA herausgegeben (d.h. das CA-Zertifikat liegt dem Client vor), kann der Client die Gültigkeit der Signatur des Server-Zertifikats mit Hilfe des Public-Keys im CA-Zertifikat prüfen. Ist die Signatur in Ordnung und das Zertifikat gültig, kann es akzeptiert werden.

Abhängig vom Browser und seiner Konfiguration wird der Benutzer von der Zertifikat-Prüfung u.U. nichts merken, wenn das CA-Zertifikat zuvor in den Browser importiert und als gültig anerkannt wurde. Liegt dem Client jedoch kein CA-Zertifikat vor, entscheidet der Benutzer, ob das Zertifikat vertrauenswürdig ist oder nicht. Der Benutzer muß dazu üblicherweise eine vom Browser initiierte Interaktion „durchklicken“, an deren Ende der Benutzer die Gültigkeit des Server-Zertifikats explizit anerkennt. Damit ist der Server authentifiziert. Anschließend werden dann Secret-Keys vereinbart, so daß die Nutz-Informationen verschlüsselt ausgetauscht werden können.
  ```

5.1 Virtuelle Server

Es ist möglich, mehrere *virtuelle Server* zu betreiben. Diese Server werden vom selben Eltern-Prozeß gestartet und können über verschiedene Namen angesprochen werden. Jeder Server kann sein eigenes Document-, CGI-Verzeichnis usw. haben. Auf diese Weise können mehrere WWW-Server auf demselben Rechner verwaltet werden.

Die Namen dieser (virtuellen) Server werden alle derselben IP-Nummer zugeordnet und müssen in einem Nameserver eingetragen sein. Der Apache wählt dann den gewünschten Server anhand des von den (meisten) Browsern mitgesendeten Servernamen aus. Mehrere SSL-Server können auf diese Weise *nicht* betrieben werden, denn bevor der Browser den Namen des gewünschten Servers sendet, erfolgt das SSL-Handshake. Zum Zeitpunkt des Handshakes sind jedoch lediglich IP- und Port-Nummern bekannt. Daher kann für **jede IP- und Port-Nummer nur ein SSL-Server aufgesetzt werden**. Daraus ergeben sich zwei Möglichkeiten, mehrere virtuelle SSL-Server zu betreiben: durch Zuordnung von mehreren IP-Nummern zum Netzwerk-Interface bzw. durch die Verwendung von mehreren Ports.

Erfolgt der Einsatz der virtuellen SSL-Server mittels unterschiedlicher IP-Nummern, werden diese IP-basiert genannt. Zu der Verwendung von anderen Ports als 443 für einen SSL-Server ist anzumerken, daß die meisten Ports unterhalb von 1024 für andere Dienste vergeben sind. Zudem muß die Port-Nummer dann in der gewünschten URL mit angegeben werden, was aus Anwendersicht nicht sehr komfortabel ist. Virtuelle SSL-Server sind also möglich, sollten aber IP-basiert sein.

5.2 Erzeugen eines Server-Zertifikats

In diesem Abschnitt werden zwei Möglichkeiten vorgestellt, ein Server-Zertifikat zu erzeugen. Die erste Möglichkeit verwendet die vom Mod-SSL-Paket zur Verfügung gestellten Hilfsmittel. Es wird ein Server-Test-Zertifikat erzeugt, welches durch eine Test-CA signiert wurde. Die zweite Variante beschreibt die Erzeugung eines Requests mit anschließender Zertifizierung durch eine reale CA. Beide Varianten verwenden das `openssl`-Binary des SSL-Pakets *OpenSSL*. Da der SSL-Apache ohne kompiliertes OpenSSL-Paket nicht zu übersetzen wäre, sollte dieses Binary vorhanden sein.

5.2.1 Erzeugen eines Zertifikats durch `make certificate`

In diesem Abschnitt beschriebene Mod-SSL-Direktiven (siehe Dokumentation des Mod-SSL-Pakets):

- `SSLCertificateFile`
- `SSLCertificateKeyFile`
- `SSLCACertificateFile`
- `SSLCACertificatePath`

Nachdem der Apache übersetzt wurde, kann durch Aufruf von `make certificate` im Apache-Quell-Verzeichnis ein Server-Zertifikat mit zugehörigem 1024-Bit-Schlüssel erstellt werden. In einzelnen Schritten („`STEP 1-4`“) werden die Angaben zur Erzeugung einer Server-Zertifikatanforderung (Serverrequest) abgefragt. Die Fragen sind sehr ausführlich und sollten keine Probleme bereiten. U.a. werden Angaben zu Land, (Bundes-)Staat usw. abgefragt. Wichtig ist die Angabe `Common Name`: hier ist unbedingt der Server-Name anzugeben, also z.B. `www.name.de`. Andernfalls werden Browser eine Warnung ausgeben, daß der Server möglicherweise nicht derjenige als der er sich ausgibt.

Im letzten Schritt („`STEP 4`“) wird nach einer Passphrase gefragt, mittels der der Private-Key des Servers geschützt werden soll. Diese Passphrase wird bei jedem Starten des Servers abgefragt, um den Private-Key entsperren zu können. (Diskussionen über Vor- und Nachteile eines verschlüsselten Server-Keys tauchen immer mal wieder auf, und auch hier wird keine „letzte“ Antwort gegeben. Klar ist, daß die Angabe einer Passphrase beim Start des Servers ein unbeaufsichtigtes Starten erschwert.)

Abschließend wird der Request zertifiziert und liegt dann im Verzeichnis `conf/ssl.crt/` unter dem Namen `server.crt` vor. Das zugehörige (von `Mod-SSL` mitgelieferte) CA-Zertifikat findet sich im Apache-Quell-Baum unter `conf/ssl.crt/snakeoil-ca-rsa.crt`. Für einen Test können die beiden Zertifikate und der Server-Key `conf/ssl.key/server.key` in das Apache-Verzeichnis kopiert werden:

- `cp conf/ssl.crt/server.crt $(APADIR)/ssl/serverCert.pem`
- `cp conf/ssl.key/server.key $(APADIR)/ssl/serverKey.pem`
- `cp conf/ssl.crt/snakeoil-ca-rsa.crt $(APADIR)/ssl/cacerts/snakeCA.pem`

Rechte und Gruppe anpassen:

- `chown root:other $(APADIR)/ssl/*.pem`
- `chmod 400 $(APADIR)/ssl/*.pem`
- `chown -R www:wwwadm $(APADIR)/cacerts`
- `chmod 440 $(APADIR)/cacerts/*`

Die entsprechenden Konfigurations-Direktiven müssen in der Konfigurationsdatei `httpd.conf` (siehe Anhang (A)) gesetzt sein, z.B.:

`SSLCertificateFile /usr/local/apache/ssl/serverCert.pem`

Die Direktive weist auf die Datei mit dem Serverzertifikat.

`SSLCertificateKeyFile /usr/local/apache/ssl/serverKey.pem`

Die Direktive weist auf die Datei mit dem Private-Key des Servers. Sollte sich der Server-Key zusammen mit dem Server-Zertifikat in einer Datei befinden, muß hier ebenfalls die Zertifikatdatei angegeben werden.

`SSLCACertificateFile /usr/local/apache/ssl/cacerts/snakeCA.pem`

Die Direktive weist auf die Datei mit dem CA-Zertifikat. CA-Zertifikate werden benötigt, um eine Client-Zertifikatprüfung durchzuführen. Es können mehrere CA-Zertifikate in *eine Datei* geschrieben werden: z.B. `cat ca1.pem ca2.pem ca3.pem > cas.pem`.

`SSLCACertificatePath /usr/local/apache/ssl/cacerts`

Die Direktive dient ebenfalls dazu, dem Server Zugriff auf CA-Zertifikate zu geben. Der Unterschied zu `SSLCACertificateFile` liegt in der Verwaltung der CA-Zertifikate. Für `SSLCACertificatePath` müssen alle Zertifikate *einzeln* in das benannte Verzeichnis kopiert werden (hier: `.../cacerts`). Anschließend werden sie dann über ihre Hash-Werte durch die Ausführung des Shell-Skripts `c_rehash` (aus dem OpenSSL-Paket, siehe auch *Das OpenSSL-Handbuch* `<http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/>`) verlinkt. Ohne diese Links findet der Apache die CA-Zertifikate nicht:

- `cd $(APACHE)/ssl/cacerts`
- `chmod 640 $(APADIR)/ssl/cacerts/*`
- `c_rehash`

5.2.2 Zertifizierung durch eine CA

Bei dieser Methode wird zunächst ein Request erzeugt, welcher anschließend durch eine CA signiert wird. Diese CA kann natürlich auch vom Betreiber des Servers realisiert werden; hierzu kann ebenfalls das OpenSSL-Paket eingesetzt werden. Auf diese Variante der Realisierung einer CA wird hier nicht weiter eingegangen.

Zur Erzeugung eines Requests muß zunächst ein Schlüsselpaar generiert werden. Mit folgendem Befehl wird ein 1024 Bit-Schlüssel erzeugt (bei Bedarf ist der OpenSSL-Pfad zu ergänzen):

```
openssl genrsa -out serverKey.pem -rand file1:file2:... 1024
```

Vorher sollte allerdings eine Umgebungsvariable `RANDFILE` gesetzt und der Zufallszahlen-Status initialisiert worden sein. Dazu kann irgendeine Datei mit (Pseudo-)Zufallsdaten nach `$(RANDFILE)` kopiert werden. Die nach der Option `-rand` angegebenen Dateien dienen als zusätzliche Zufallsdaten für die Schlüsselerzeugung.

Das `genrsa`-Kommando erzeugt ein unverschlüsseltes Schlüsselpaar, der Private-Key ist also von jedem verwendbar. Ist eine Verschlüsselung des Private-Keys gewünscht, kann bei obigem Kommando zusätzlich die Option `-des3` angegeben werden.

Die Erzeugung des Requests erfolgt anschließend durch folgenden Befehl:

```
openssl req -new -inkey serverKey.pem -out severReq.pem
```

Nach Eingabe des Befehls müssen folgende Angaben gemacht werden (beispielhafte Eingaben sind in doppelten Anführungsstrichen):

```
Using configuration from /usr/local/openssl/openssl.cnf
Enter PEM pass phrase:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:"DE"
State or Province Name (full name) [Some-State]:"Schleswig-Holstein"
Locality Name (eg, city) []:"Kiel"
Organization Name (eg, company) [Internet Widgits Pty Ltd]:"WWW UnLtd"
Organizational Unit Name (eg, section) []:"Abt. Web-Server"
Common Name (eg, YOUR name) []:"www.www-unltd.de"
Email Address []:"wwwadmin@www-unltd.de"

Please enter the following "extra" attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

(Für den genauen Inhalt muß die zertifizierende CA bzw. deren Policy konsultiert werden.)

Achtung:
Soll der signierte Request später als SSL-Server-Zertifikat verwendet werden, muß als `Common Name` unbedingt der Server-Name angegeben werden!

Die beiden letzten Angaben (`challenge password` und `optional company name`) tauchen als Klartext im Attribut-Bereich des Requests auf. Soll später ein Zertifikat zurückgerufen werden, kann sich der Inhaber gegenüber der CA, auch bei Verlust des Private-Keys, durch Angabe dieser Felder als Inhaber „ausweisen". Die Verwaltung dieser Angaben kann von der unterzeichnenden CA durchgeführt werden. Ob die signierende CA diese Felder unterstützt, sollte mit der CA geklärt werden.

Der so erzeugte Request (die Datei `serverReq.pem`) kann dann einer CA zum Signieren vorgelegt werden. Alternativ kann auch eine eigene (Test-)CA installiert und der Request signiert werden. Auf diese Möglichkeit wird hier nicht weiter eingegangen.

Liegt das Server-Zertifikat später vor, werden das Zertifikat und der Private-Key nach `$(APADIR)/ssl` kopiert.

Für die Konfiguration der Mod-SSL-Direktiven gelten die im obigen Abschnitt (5.2.1) gemachten Anmerkungen.

5.3 Prüfung von Client-Zertifikaten

In diesem Abschnitt beschriebene Mod-SSL-Direktiven (siehe Dokumentation des Mod-SSL-Pakets):

- `SSLVerifyClient`
- `SSLVerifyDepth`
- `SSLRequire`
- `SSLOptions`
- `SSLRequireSSL`

Ist zusätzlich zu der obligatorischen Server-Authentifizierung (siehe obigenr Abschnitt (5)) noch eine Client-Authentifizierung erforderlich, schickt der Client sein Zertifikat während des SSL-Handshakes an den Server. Der Server sendet dazu dem Client eine zufällige Nachricht, die dieser dann signiert zusammen mit seinem Zertifikat an den Server sendet. Dieser kann dann die Signatur mit dem Public-Key aus dem Client-Zertifikat überprüfen. Das Client-Zertifikat selbst wird dann ebenfalls überprüft, wozu das Zertifikat der Herausgeber-CA erforderlich ist.

SSLVerifyClient

Der SSL-Server kann in Bezug auf die Client-Authentifizierung auf vier verschiedene Arten konfiguriert werden (Schlüsselwort `SSLVerifyClient`):

`none`

Es ist kein Zertifikat erforderlich, es erfolgt also keine Client-Authentisierung.

`optional`

Ein gültiges Zertifikat wird akzeptiert, ist aber nicht erforderlich.

`require`

Ein gültiges Zertifikat muß vorgelegt werden. Außerdem muß es überprüfbar sein, der Server muß also das Zertifikat der Unterzeichner-CA vorliegen haben.

`optional_no_ca`

Ein gültiges Zertifikat kann vorgelegt werden, muß aber nicht überprüfbar sein.

Für einen realen Betrieb kommen `none` oder `require` in Frage. Die anderen Werte können für einen Test-Server interessant sein.

Durch Setzen der `SSLVerifyClient`-Direktive auf `require` erhält ein Client erst nach erfolgter Authentisierung Zugriff auf die HTML-Seiten eines Servers. Es ist aber auch möglich, dem Client Zugriff auf das Document-Root-Verzeichnis ohne Zertifikat zu gewähren (`SSLVerifyClient none`) und dann den Zugriff auf Seiten in einem Unterverzeichnis erst nach erfolgreicher Client-Authentisierung zuzulassen. Dazu wird eine zweite `SSLVerifyClient`-Direktive innerhalb einer `Directory`-Anweisung für dieses Unterverzeichnis auf `require` gesetzt. Der Ablauf ist dann folgender: der Client verbindet sich zum Server, und es wird zunächst eine Server-Authentifizierung durchgeführt. Danach hat der Client Zugriff auf das Document-Root-Verzeichnis. Versucht dann der Client, von den nicht geschützten Seiten auf die Seiten in dem durch `SSLVerifyClient require` geschützten Verzeichnis zuzugreifen, werden die Bedingungen für die SSL-Verbindung erneut ausgehandelt (*Renegotiation*). Der Server fordert dann vom Client ein Zertifikat und gestattet den Zugriff erst nach erfolgreicher Authentisierung des Clients.

SSLVerifyDepth

Zusammen mit der Direktive `SSLVerifyClient` wird die Direktive `SSLVerifyDepth` n eingesetzt. Durch die Zahl n wird die maximale Länge der zu prüfenden Zertifikatkette festgelegt. Ein Wert von n=`1` bedeutet, daß der Client direkt von einer Root-CA signiert wurde. Ein Wert von 3 erlaubt die Prüfung einer Zertifikatkette aus drei CA-Zertifikaten und Client-Zertifikat.

SSLRequire, **SSLOptions**, **SSLRequireSSL**

Wie das Client-Zertifikat auszusehen hat, kann sehr fein durch die `SSLRequire`-Direktive kontrolliert werden. Der SSL-Server setzt, wenn die Direktive `SSLOptions` die Option `+ExportCertData` enthält, mehrere Umgebungsvariablen, deren Werte aus den einzelnen Zertifikatfeldern bestehen. Beispielweise kann die Variable `SSL_CLIENT_S_DN_L` den Wert `Kiel` haben, also den Inhalt des Locality-Feldes (L) aus dem Distinguished Name (DN) des Zertifikat-Subjects (S). Die Bezeichnungen der anderen Variablen bauen sich analog auf. Ein vollständige Liste der Namen der gesetzten Variablen kann der Mod-SSL-Dokumentation entnommen werden.
Das oben bei der `SSLVerifyClient`-Direktive angeführte Beispiel des durch Renegotiation geschützten Unterverzeichnisses kann durch Verwendung dieser Variablen weiter verfeinert werden: z.B. sollen nur Clients mit überprüfbarem Zertifikat Zugriff auf das Unterverzeichnis bekommen, wenn das Zertifikat im Locality-Feld des DN die Zeichenkette `Kiel` enthält. Die entsprechende Direktive zur Prüfung des Locality-Feldes sieht so aus:

```
SSLOptions  +ExportCertData +OptRenegotiate +StrictRequire
SSLRequire  %{SSL_CLIENT_S_DN_L} eq "Kiel"
```

Die Option `+StrictRequire` zu setzen, hat eine Sicherungsfunktion. Wenn unter den nicht-SSL-Direktiven des Apache die Direktive `Satisfy` auf `any` gesetzt ist, kann ein Client einen ungewollten Zugriff auf das Unterverzeichnis bekommen. Bei gleichzeitiger Verwendung von `SSLRequire` und `SSLRequireSSL` erhält ein Client Zugang, sobald eine der Direktiven den Zugang zuläßt, aber die andere Direktive aber nicht. Durch Setzen von `+StrictRequire` wird der Zugriff verweigert, auch wenn eine der beiden Direktiven es zulassen würde. Die `Satisfy any`-Direktive ist somit in diesem Fall unwirksam.

5.4 Widerrufslisten (CRLs)

In diesem Abschnitt beschriebene Mod-SSL-Direktiven:

- `SSLCARevocationPath`
- `SSLCARevocationFile`

Das `mod_ssl`-Paket unterstützt die Verwendung von Widerrufslisten (CRL) mit dem Apache. Eine CRL enthält die Seriennummern zurückgerufener Zertifikate und den Namen der CA, die die CRL herausgegeben hat. Wird eine der folgenden Direktiven benutzt, werden Client-Zertifikate bei aktivierter Client-Authentisierung anhand einer CRL geprüft. Ist ein Client-Zertifikat widerrufen worden, wird der Zugriff dieses Client abgelehnt. CRLs werden in regelmäßigen Abständen (z.B. wöchentlich oder monatlich) von den CAs herausgegeben. CRLs sind keine differentiellen Listen, sondern enthalten immer alle von einer CA widerrufenen Zertifikate. Die alte CRL einer CA kann also durch die neu herausgegebene CRL ersetzt werden. Trotzdem kann es sinnvoll sein, alte CRLs aufzuheben (s.u.). Für jede unter den `SSLCACertificate`...-Direktiven (s.o.) angegebene CA sollte die entsprechende CRL dem Server zur Verfügung stehen.

SSLCARevocationPath, SSLCARevocationFile

Die CRLs werden dem Apache über eine der folgenden Direktiven (analog den `SSLCACertificate`-Direktiven) zur Verfügung gestellt, z.B.:

`SSLCARevocationFile /usr/local/apache/ssl/crl.crl`

Die Direktive weist auf eine Datei mit der CRL. Liegen CRLs von verschiedenen CAs vor, können alle CRLs in *eine Datei* geschrieben werden: z.B. `cat ca1.crl ca2.crl ca3.crl > crl.crl`.

`SSLCARevocationPath /usr/local/apache/ssl/crls`

Die Direktive hat dieselbe Funktion wie `SSLCARevocationFile`. Die Unterschied liegt in der Verwaltung der CRLs. Für `SSLCARevocationPath` müssen alle Zertifikate *einzeln* in das benannte Verzeichnis kopiert werden (hier: `.../ssl/crls`). Anschließend müssen die Listen dann über ihre Hash-Werte verlinkt werden. Bei mehreren Listen läßt sich das am einfachsten durch das unten stehende Shell-Skript erledigen.

CAs geben üblicherweise CRLs heraus, die immer sämtliche aktuell widerrufenen Zertifikate enthalten. Daher kann die alte CRL einer CA gelöscht oder besser für Archivierungszwecke umbenannt werden. Das folgende Shell-Skript löscht zunächst den Hash-Link auf die alte CRL und benennt dann die alte CRL um, indem das aktuelle Datum angehängt wird. Anschließend wird die neue CRL verschoben und verlinkt. Im Beispiel wird angenommen, daß die neue CRL unter `/tmp/bsp.crl` vorliegt und im Verzeichnis `/usr/local/apache/ssl/crls` installiert werden soll.

```
#!/bin/sh
cd /usr/local/apache/ssl/crls
rm `openssl crl -in /tmp/bsp.crl -hash -noout`.r0
mv bsp.crl bsp.crl.`date '+%m%d%y'`
mv /tmp/bsp.crl .
ln -s $i `openssl crl -in bsp.crl -hash -noout`.r0
chmod 644 bsp.crl.*
```

Die Verwaltung der CRLs ist mit einem gewissen Aufwand verbunden, da CRLs von CAs u.U. monatlich aktualisiert werden und entsprechend die Listen auf dem SSL-Server zu warten sind. Möglicherweise läßt sich der Vorgang durch ein entsprechendes Skript per Cron-Job automatisieren. Sollte die CRL nicht (wie im Skript angenommen) im PEM-Format vorliegen, sondern im DER-Format, muß in dem Skript in der zweiten Zeile (nach `openssl crl`) noch die Option `-inform der` eingefügt werden.

5.5 Starten und Stoppen des Apache

Der Server kann über das Skript `apachectl` oder durch Eingabe eines der folgenden Kommandos gestartet bzw. gestoppt werden:

- `/usr/local/bin/httpd`
  In der Beispiel-Konfigurationsdatei im Anhang (A) sind die SSL-bezogenen Teile der Apache-Konfiguration über `<IfDefine SSL>...</IfDefine SSL>` gekapselt. Somit startet dieses Kommando den Apache *ohne* SSL-Funktionalität.
- `/usr/local/bin/httpd -DSSL`
  Mit diesem Kommando wird der Apache zusätzlich *mit* SSL-Funktionalität gestartet.
- `kill -TERM `cat /usr/local/apache/logs/httpd.pid``
  Mit diesem Kommando wird der Server gestoppt.

Nach Eingabe des zweiten Kommandos wird der Schlüssel für den SSL-Server geladen. Liegt dieser verschlüsselt vor, wird nach einer Passphrase gefragt. Nach Eingabe der Passphrase startet der Server dann. Das impliziert, daß beim Starten des SSL-Servers jemand zugegen sein muß. Alternativ kann die Passphrase in einer - wie auch immer beschaffenen - Datei abgelegt werden. Wichtig ist nur, daß die Passphrase von dieser Datei bzw. dem Programm auf der Standardausgabe ausgegeben wird. Dazu muß die SSL-Direktive `SSLPassPhraseDialog` auf den Pfad zu dieser Datei gesetzt werden:

```
SSLPassPhraseDialog /usr/local/bin/gibpassphrase
```

Voreinstellung der Direktive ist `builtin`, was für die Eingabe der Passphrase am Prompt steht. Bei einem unbeaufsichtigten Start des Servers (z.B. nach einem Reboot) kann mit dem obigen ersten Kommando (per init-Skript) bei verschlüsselter Passphrase zumindest der Nicht-SSL-Server wieder in Betrieb genommen werden. Oder es kann alternativ die Passphrase-Datei eingesetzt werden.

Das Skript `apachectl` nutzt letztendlich die obigen drei Kommandos. Die Funktion der obigen Kommandos würde mit dem Skript durch folgende Aufrufe erreicht:

- `/usr/local/bin/apachectl`
- `/usr/local/bin/apachectl startssl`
- `/usr/local/bin/apachectl stop`

Das Skript bietet noch eine Reihe weiterer Funktionen, allerdings z.T. in Abhängigkeit von der Server-Konfiguration. Für weitere Informationen wird auf das kommentierte Skript verwiesen.

A `http.conf` - Beispiel-Datei

```
# The configuration directives are grouped into three basic sections:
#  1. Directives that control the operation of the Apache server process as a
#     whole (the 'global environment').
#  2. Directives that define the parameters of the 'main' or 'default' server,
#     which responds to requests that aren't handled by a virtual host.
#     These directives also provide default values for the settings
#     of all virtual hosts.
#  3. Settings for virtual hosts, which allow Web requests to be sent to
```

```
#     different IP addresses or hostnames and have them handled by the
#     same Apache server process.



# ServerType legt fest, wie der Server gestartet wird. Moeglich sind
# "inetd" oder "standalone". Mit inetd wird der Server von inetd ueber einen
# Eintrag in inetd.conf gestartet. Wird der Server auf diese Weise gestartet,
# wird fuer jeden HTTP-Request ein neuer Server gestartet und anschliessend
# beendet. Die Folge ist, dass der Rechner stark belastet werden kann. Und
# ausserdem:
# SSL Servers MUST be standalone, currently.

ServerType standalone


# ServerRoot: The directory the server's config, error, and log files
# are kept in

ServerRoot "/usr/local/apache"


# PidFile: The file the server should log its pid to

PidFile logs/httpd.pid


# ScoreBoardFile: File used to store internal server process information.
# Not all architectures require this.  But if yours does (you'll know because
# this file will be  created when you run Apache) then you *must* ensure that
# no two invocations of Apache share the same scoreboard file.

# ScoreBoardFile logs/apache_runtime_status


# Saemtliche Server-Direktiven befinden sich in httpd.conf. Daher:

ResourceConfig /dev/null
AccessConfig /dev/null


# Nach wieviel Sekunden die Verbindung geschlossen wird.

Timeout 300

# Ermoeglicht es, mehrere Anfragen ueber eine TCP-Verbindung abzusetzen.

KeepAlive On
```

```
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited amount.
# We recommend you leave this number high, for maximum performance.

MaxKeepAliveRequests 100


# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.

KeepAliveTimeout 15


# Server-pool size regulation.  Rather than making you guess how many
# server processes you need, Apache dynamically adapts to the load it
# sees --- that is, it tries to maintain enough server processes to
# handle the current load, plus a few spare servers to handle transient
# load spikes (e.g., multiple simultaneous requests from a single
# Netscape browser).
#
# It does this by periodically checking how many servers are waiting
# for a request.  If there are fewer than MinSpareServers, it creates
# a new spare.  If there are more than MaxSpareServers, some of the
# spares die off.  The default values are probably OK for most sites.

MinSpareServers 5
MaxSpareServers 10


# Number of servers to start initially --- should be a reasonable ballpark
# figure.

StartServers 5


# Limit on total number of servers running, i.e., limit on the number
# of clients who can simultaneously connect --- if this limit is ever
# reached, clients will be LOCKED OUT, so it should NOT BE SET TOO LOW.
# It is intended mainly as a brake to keep a runaway server from taking
# the system with it as it spirals down...

MaxClients 150


# MaxRequestsPerChild: the number of requests each child process is
# allowed to process before the child dies.  The child will exit so
# as to avoid problems after prolonged use when Apache (and maybe the
# libraries it uses) leak memory or other resources.  On most systems, this
# isn't really needed, but a few (such as Solaris) do have notable leaks
# in the libraries.
```

```
#
# NOTE: This value does not include keepalive requests after the initial
#       request per connection. For example, if a child process handles
#       an initial request and 10 subsequent "keptalive" requests, it
#       would only count as 1 request towards this limit.
#

MaxRequestsPerChild 10000


# Die Port-Direktive legt fest, an welchem Port der Server horcht. Sind
# zusaetzlich
# noch Listen oder BindAddress gesetzt, hat die Direktive keine Wirkung. Soll
# allerdings der Default-Port ein anderer als 80 sein, muss der Port ueber
# die Port-Anweisung gesetzt sein.
# Port setzt ausserdem die Umgebungs-Variable "SERVER_PORT", die fuer CGI und
# SSI benoetigt wird.
# The default port for SSL is 443...

Port 80



##############################################################################
##############################################################################
#
# Die folgenden Zeilen mit den LoadModule- und AddModule-Direktiven sind fuer
# einen DSO-Apache gedacht. Kommt ein Nicht-DSO-Server zum Einsatz, sollten die
# Zeilen bis zu der Zeile   "# Ende der DSO-relevanten Direktiven"
# auskommentiert oder geloescht werden.
#
#
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built as a DSO you
# have to place corresponding `LoadModule' lines at this location so the
# directives contained in it are actually available _before_ they are used.
# Please read the file README.DSO in the Apache 1.3 distribution for more
# details about the DSO mechanism and run `httpd -l' for the list of already
# built-in (statically linked and thus always available) modules in your httpd
# binary.
#
# Note: The order is which modules are loaded is important.  Don't change
# the order below without expert advice.
#
# Example:
# LoadModule foo_module libexec/mod_foo.so
LoadModule mmap_static_module   libexec/mod_mmap_static.so
LoadModule vhost_alias_module   libexec/mod_vhost_alias.so
LoadModule env_module           libexec/mod_env.so
```

```
# LoadModule define_module      libexec/mod_define.so
LoadModule config_log_module    libexec/mod_log_config.so
# LoadModule agent_log_module   libexec/mod_log_agent.so
# LoadModule referer_log_module libexec/mod_log_referer.so
# LoadModule mime_magic_module  libexec/mod_mime_magic.so
LoadModule mime_module          libexec/mod_mime.so
LoadModule negotiation_module   libexec/mod_negotiation.so
# LoadModule status_module      libexec/mod_status.so
# LoadModule info_module        libexec/mod_info.so
# LoadModule includes_module    libexec/mod_include.so
# LoadModule autoindex_module   libexec/mod_autoindex.so
LoadModule dir_module           libexec/mod_dir.so
LoadModule cgi_module           libexec/mod_cgi.so
# LoadModule asis_module        libexec/mod_asis.so
# LoadModule imap_module        libexec/mod_imap.so
LoadModule action_module        libexec/mod_actions.so
# LoadModule speling_module     libexec/mod_speling.so
# LoadModule userdir_module     libexec/mod_userdir.so
LoadModule alias_module         libexec/mod_alias.so
# LoadModule rewrite_module     libexec/mod_rewrite.so
LoadModule access_module        libexec/mod_access.so
# LoadModule auth_module        libexec/mod_auth.so
# LoadModule anon_auth_module   libexec/mod_auth_anon.so
# LoadModule dbm_auth_module    libexec/mod_auth_dbm.so
# LoadModule digest_module      libexec/mod_digest.so
# LoadModule proxy_module       libexec/libproxy.so
# LoadModule cern_meta_module   libexec/mod_cern_meta.so
# LoadModule expires_module     libexec/mod_expires.so
# LoadModule headers_module     libexec/mod_headers.so
# LoadModule usertrack_module   libexec/mod_usertrack.so
# LoadModule example_module     libexec/mod_example.so
# LoadModule unique_id_module   libexec/mod_unique_id.so
LoadModule setenvif_module      libexec/mod_setenvif.so

<IfDefine SSL>
    LoadModule ssl_module           libexec/libssl.so
</IfDefine>



#  Reconstruction of the complete module list from all available modules
#  (static and shared ones) to achieve correct module execution order.
#  [WHENEVER YOU CHANGE THE LOADMODULE SECTION ABOVE UPDATE THIS, TOO]

ClearModuleList
AddModule       mod_mmap_static.c
AddModule       mod_vhost_alias.c
AddModule       mod_env.c
# AddModule     mod_define.c
AddModule       mod_log_config.c
```

```
# AddModule     mod_log_agent.c
# AddModule     mod_log_referer.c
# AddModule     mod_mime_magic.c
AddModule       mod_mime.c
AddModule       mod_negotiation.c
# AddModule     mod_status.c
# AddModule     mod_info.c
# AddModule     mod_include.c
# AddModule     mod_autoindex.c
AddModule       mod_dir.c
AddModule       mod_cgi.c
# AddModule     mod_asis.c
# AddModule     mod_imap.c
AddModule       mod_actions.c
# AddModule     mod_speling.c
# AddModule     mod_userdir.c
AddModule       mod_alias.c
# AddModule     mod_rewrite.c
AddModule       mod_access.c
# AddModule     mod_auth.c
# AddModule     mod_auth_anon.c
# AddModule     mod_auth_dbm.c
# AddModule     mod_digest.c
# AddModule     mod_proxy.c
# AddModule     mod_cern_meta.c
# AddModule     mod_expires.c
# AddModule     mod_headers.c
# AddModule     mod_usertrack.c
# AddModule     mod_example.c
# AddModule     mod_unique_id.c
AddModule       mod_so.c
AddModule       mod_setenvif.c

<IfDefine SSL>
    AddModule mod_ssl.c
</IfDefine>

# Ende der DSO-relevanten Direktiven
#
##############################################################################
##############################################################################



# Hat der Server mehrere Interfaces, horcht er auf jedem Interface, auf den bei
# "Port" angegebenen Port. Es ist auch moeglich, verschiedene IP-Adressen
# anzugeben (soweit der Rechner entsprechend konfiguriert ist). Ebenso sind
# unterschiedliche Hostnames moeglich. Listen hat hoehere
# Prioritaet als Port und BindAdress.
```

```
Listen 127.0.0.42:80


<IfDefine SSL>
    Port   443
    Listen 127.0.0.42:443
</IfDefine>


# User- und Group-Direktive setzen die UserID bzw. Group-ID, unter der der
# Server laeuft. Am besten einen eigenen User und Group einrichten und Server
# als root starten. Nach dem Start wechselt der Server auf den weniger
# priviligierten User-/Group-ID.

User www
Group wwwadm


# An wen geht eine Meldung im Fehlerfall, z.B. steht diese Adresse
# auf vom Server generierten Seiten mit Fehlermeldungen.

ServerAdmin wwwadmin@name.de


# ServerName allows you to set a host name which is sent back to clients for
# your server if it's different than the one the program would get (i.e., use
# "www" instead of the host's real name).
#
# Note: You cannot just invent host names and hope they work. The name you
# define here must be a valid DNS name for your host. If you don't understand
# this, ask your network administrator.
# If your host doesn't have a registered DNS name, enter its IP address here.
# You will have to access it by its address (e.g., http://123.45.67.89/)
# anyway, and this will make redirections work in a sensible way.
#
ServerName www.name.de


# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.

DocumentRoot "/var/www/www80"


# Durch folgende Anweisung wird Default-Zugriff auf das gesamte Datei-System
# des Rechners abgeschaltet, sowie alle Includes und .htaccess-overrides.
# Bei Bedarf kann der Zugriff fuer einzelne Verzeichnisse durch eine neue
# Directory-Anweisung wieder eingeschaltet werden.
# Directory bezieht sich auf den absoluten Pfad.
```

```
# Die Anweisungen Directory, Files, Location, bilden eine Hirarchie mit in
# dieser Reihenfolge zunehmender Prioritaet. Die Angabe von
# Verzeichnissen kann fuer alle drei Direktiven auch durch Wildcards und
# Regulaere Ausdruecke erfolgen.

<Directory />
    order deny,allow
    #  Zugriff aus allen Domains verbieten
    deny from all
    #  ExecCGI, FollowSymLinks, Includes, Indexes, Multiview abschalten
    Options none
    #  .htaccess-Dateien ignorieren
    AllowOverride none
</Directory>


# Wegen der oben erwaehnten Prioritaets-Hierarchie muss hier eine Directory-D
# verwendet werden, obwohl eine "Location /" dasselbe bezeichnet.

<Directory /var/www/www80>

    # Nach "deny from all" in der "Directory /"-Anweisung wird hier wieder der
    # Zugriff gestattet. Der Zugriff sollte auch gestattet werden, damit
    # bei einer Standard-Anfrage der Default "index.html" zurueckgeliefert
    # werden kann. In darauffolgenden "Directory"-Anweisungen kann der
    # Zugriff wieder auf HTML-Dateien beschraenkt werden.

    Order allow,deny
    allow from all

</Directory>


# Fuer alle Unterverzeichnisse Zugriff verbieten.
# Muss Directory statt Location sein, da sonst die Beschraenkung mit deny
# nicht mehr ueber Files fuer HTML-docs gemildert werden koennte (Location hat
# hoehere Prioritaet als Files).

<Directory /var/www/www80/*/>
    order deny,allow
    deny from all
</Directory>


# Durch folgende Anweisungen wird der Name der Index-Datei festgelegt

DirectoryIndex index.html


# UseCanonicalName:  (new for 1.3)  With this setting turned on, whenever
```

```
# Apache needs to construct a self-referencing URL (a URL that refers back
# to the server the response is coming from) it will use ServerName and
# Port to form a "canonical" name.  With this setting off, Apache will
# use the hostname:port that the client supplied, when possible.  This
# also affects SERVER_NAME and SERVER_PORT in CGI scripts.

UseCanonicalName On


# TypesConfig describes where the mime.types file (or equivalent) is
# to be found.
# Die neuen, SSL-bezogenen MIMI-Types werden weiter unten im SSL-Teil
# des Servers mit der Direktive "AddType" hinzugefuegt. Somit ist
# keine Aenderung von "mime.types" noetig.

TypesConfig conf/mime.types


# Default Mime-Type, wenn kein passender Eintrag gefunden wurde.

DefaultType text/plain


# "Off" in der folgenden Direktive schreibt die IP-Nummern der Clients statt
# deren Namen in die Log-Dateien. Andernfalls werden die Namen per
# DNS-Lookup geholt, was sich negativ auf die Performanz des Servers
# auswirkt.

HostnameLookups Off


# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# container, error messages relating to that virtual host will be
# logged here.  If you *do* define an error logfile for a <VirtualHost>
# container, that host's errors will be logged there and not here.

ErrorLog logs/80error.log


# LogLevel: Control the number of messages logged to the error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.

LogLevel warn


# The following directives define some format nicknames for use with
# a CustomLog directive (see below).
#
```

```
# Custom logging
# Datum/Zeit (t), 1. Zeile der Client-Anfrage(r), Http-Statuscode(s), Groesse
# der gelieferten Datei (b), Protokollierung von Http-Headern (i)

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent


# If you would like to have agent and referer logfiles, uncomment the
# following directives.
#
#CustomLog logs/referer_log referer
#CustomLog logs/agent_log agent


# If you prefer a single logfile with access, agent, and referer information
# (Combined Logfile Format) you can use the following directive.

CustomLog logs/80access.log combined


# Optionally add a line containing the server version and virtual host
# name to server-generated pages (error documents, FTP directory listings,
# mod_status and mod_info output etc., but not CGI generated documents).
# Set to "EMail" to also include a mailto: link to the ServerAdmin.
# Set to one of:  On | Off | EMail

ServerSignature Off


# Alias fuer die Icons

Alias /icons/ /usr/local/apache/icons/


# Folgendes kann als Location-Direktive gesetzt werden (da innerhalb
# DocumentRoot und hoehere Prioritaet als Directory).

<Location /icons/*.gif>
    Order deny,allow
    allow from all
</Location>


<Location /icons/*.jpg>
    Order deny,allow
    allow from all
</Location>
```

```
# CGI-Skripte sollten nicht im Dokument-Root stehen, werden aber relativ dazu
# verwendet. Daher

ScriptAlias /cgi/ /usr/local/apache/cgi-bin/


<Files sample.cgi>

    # Setzen des cgi-Handlers auf genau eine Datei. Durch mehrere
    # Files-Anweisungen entsprechend auf mehrere Dateien. Etwas Paranoid,
    # alternativ kann jede auf cgi endende Datei "freigeschaltet" werden.

    SetHandler cgi-script
    order deny,allow
    allow from all

</Files>


# Die beiden Borwser-Typen (Mozilla/2 gilt sowohl fuer Navigator 2 wie
# auch fuer aeltere MSIE-Versionen, die sich als Navigator ausgegeben
# haben) haben Probleme mit persistenten Verbindungen. Daher keine
# persistenten Verbindungen fuer diese Browser ("noekeepalive").
# Ausserdem hat der MSIE Probleme mit HTTP/1.1-Antworten. Die Anfragen
# kommen zwar in HTTP/1.1 vom MSIE; durch die "downgrade-1.0"-Option wird dem
# Apache allerdings eine HTTP/1.0-Anfrage vorgetaeuscht, so dass die
# Antwort in HTTP/1.0 erfolgt und die Probleme mit MSIE vermieden werden.

BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0

# Schwierigkeiten mit dem von mod_rewrite eingefuegten
# "Vary"-Headern. Daher keine solche Header bei folgendem Browser-Typ
# einfuegen.

# BrowserMatch "MSIE 4\.0" force-no-vary

# The following directive disables HTTP/1.1 responses to browsers which
# are in violation of the HTTP/1.0 spec by not being able to grok a
# basic 1.1 response.
# Der Apache signalisiert in seinen Antworten, dass er ein HTTP/1.1
# faehiger Server ist. Einige Clients interpretieren dass
# faelschlicherweise Angabe zum Protokoll der Anworten und verweigern
# die Annahme der Antwort. Daher wird diesen Clients mit folgenden
# Direktiven ein HTTP/1.0 angeboten...
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
```

```
################################################################################
################################################################################
##
##  SSL Global Context
##
##  All SSL configuration in this context applies both to
##  the main server and all SSL-enabled virtual hosts.
##

<IfDefine SSL>

    # Neue SSL-relevante Mime-Types hinzufuegen.

    AddType application/x-pkcs7-crl        crl
    AddType application/x-x509-ca-cert     cacrt
    AddType application/x-x509-email-cert  emailcrt
    AddType application/x-x509-user-cert   usercrt

</IfDefine>


<IfModule mod_ssl.c>

    #   Pass Phrase Dialog:
    #   Configure the pass phrase gathering process.
    #   The filtering dialog program (`builtin' is a internal
    #   terminal dialog) has to provide the pass phrase on stdout.

    SSLPassPhraseDialog      builtin


    #   Inter-Process Session Cache:
    #   Configure the SSL Session Cache: First either `none'
    #   or `dbm:/path/to/file' for the mechanism to use and
    #   second the expiring timeout (in seconds).
    #
    #   "shm" benoetigt die MM-Bibliothek.

    # SSLSessionCache        none
    SSLSessionCache          shm:logs/ssl_scache(512000)
    # SSLSessionCache        dbm:logs/ssl_scache
    SSLSessionCacheTimeout   300


    #   Semaphore:
    #   Configure the path to the mutual explusion semaphore the
    #   SSL engine uses internally for inter-process synchronization.
```

```
    # SSLMutex   file:logs/ssl_mutex
    SSLMutex     sem

    #   Pseudo Random Number Generator (PRNG):
    #   Configure one or more sources to seed the PRNG of the
    #   SSL library. The seed data should be of good random quality.
    #
    #   Leider kein /dev/random unter Solaris. Es gibt aber die Moeglichkeit,
    #   ueber externe Programme Zufallsdaten zuzufuehren (z.B. "truerand" im
    #   Mod-SSL-Quell-Verzeichnis unter "pkg.contrib")
    # SSLRandomSeed startup exec:/usr/local/bin/truerand 16
    #   Ohne "truerand":

    SSLRandomSeed startup builtin
    SSLRandomSeed connect builtin


    #   Logging:
    #   The home of the dedicated SSL protocol logfile. Errors are
    #   additionally duplicated in the general error log file.  Put
    #   this somewhere where it cannot be used for symlink attacks on
    #   a real server (i.e. somewhere where only root can write).
    #   Log levels are (ascending order: higher ones include lower ones):
    #   none, error, warn, info, trace, debug.

    SSLLog      logs/ssl_engine.log
    SSLLogLevel info

</IfModule>



<IfDefine SSL>

    ##
    ## SSL Virtual Host Context
    ##

    #  Grundsaetzlich gilt, dass jeder virtuelle Host die Werte der im
    #  "Hauptserver" gesetzten Direktiven "erbt".


    <VirtualHost www.name.de:443>

        #  Im Document-Root-Verzeichnis wird auf eine
        #  Client-Authentifizierung verzichtet.

        #  General setup for the virtual host

        DocumentRoot    /var/www/www443
```

```
ServerName      www.name.de
ServerAdmin     wwwadm@name.de
ErrorLog        logs/443error.log
CustomLog       logs/443access.log \
                "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"

SSLVerifyClient none
SSLEngine       on
SSLProtocol     all -SSLv2

#  SSL Cipher Suite: Ueber diese Direktive kann festgelegt werden,
#  welche Krypto-Algorithmen und in welcher Prioritaet diese verwendet
#  werden.

SSLCipherSuite ALL:!ADH:!SSLv2:!eNULL:RC4+RSA:+HIGH:+MEDIUM:+LOW:+EXP

SSLCertificateFile        /usr/local/apache/ssl/serverCert.pem
SSLCertificateKeyFile     /usr/local/apache/ssl/serverKey.pem
# SSLCertificateChainFile /usr/local/apache/ssl/cacerts/cachain.pem

SSLCACertificatePath      /usr/local/apache/ssl/cacerts
# SSLCACertificateFile    /usr/local/apache/ssl/CAcerts.pem

SSLCARevocationPath       /usr/local/apache/ssl/crls
# SSLCARevocationFile     /usr/local/apache/ssl/crls.pem


<Directory /var/www/www443>

    # Nach "deny from all" in der "Directory /"-Anweisung wird hier
    # wieder der Zugriff gestattet.

    allow from all

</Directory>


<Directory /var/www/www443/*/>
    order deny,allow
    deny  from all
</Directory>


<Directory /var/www/www443/restricted>

    # Auf Dateien in diesem Verzeichnis sollen nur Clients mit
    # gueltigem Zertifikat zugreifen koennen. Das erfordert u.U.
    # ein erneutes SSL-Handshake ("Renegotiation"), da eine SSL-Session
    # besteht, der Client aber noch nicht authentifiziert wurde.
```

```
        SSLVerifyClient  require
        SSLVerifyDepth   5
        SSLOptions       +ExportCertData +StrictRequire +OptRenegotiate


        # Verfeinerte Zugriffssteuerung ueber Zertifikat-Attribute.
        # Hat nur indirekt etwas mit der x509-basierten Zugriffskontrolle
        # zutun, da nicht nur die Gueltigkeit des Client-Zertifkats
        # Vorraussetzung ist, sondern das Zertifikat gewisse
        # (DN-)Eigenschaften besitzen muss.
        # Vom Prinzip her eher mit einer Passwort-gesteuerten
        # Zugriffskontrolle vergleichbar.

        SSLRequire       %{SSL_CLIENT_S_DN_C} eq "DE" \
                         and %{SSL_CLIENT_S_DN_L} eq "Kiel" \
                         and %{SSL_CLIENT_S_DN_O} eq "WWW UnLtd"

        allow from all

    </Directory>

    # Wieder mal Browser-"Macken" abfangen.
    SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown

  </VirtualHost>

</IfDefine>
```

B Über dieses Dokument

Das SSL-Apache Handbuch

Dieses Dokument wurde zusammengestellt von den Mitarbeitern der *DFN-PCA* mit maßgeblicher Unterstützung durch Lars Weber (*3weber@informatik.uni-hamburg.de*).

C Links zu der dokumentierten Software

- **OpenSSL**: *http://www.openssl.org/*
  (*DFN-PCA Mirror* <ftp://ftp.pca.dfn.de/pub/tools/net/openssl/>)
- Obsolet: **SSLeay**: *http://www.ssleay.org/*
  (*DFN-PCA Mirror* <ftp://ftp.pca.dfn.de/pub/tools/net/ssleay/>)
- Obsolet: **pkcs12**: *http://www.drh-consultancy.demon.co.uk/pkcs12faq.html*
- **pfx**: *http://www.drh-consultancy.demon.co.uk/pkcs12faq.html*
- Obsolet: **ca-fix**: *http://www.drh-consultancy.demon.co.uk/ca-fix.html*
- **Apache-SSL**: *http://www.apache-ssl.org/*
  (*DFN-PCA Mirror* <ftp://ftp.pca.dfn.de/pub/tools/net/sslapache/>)

- **mod_ssl**: *http://www.modssl.org/*
  (*DFN-PCA Mirror* `<ftp://ftp.pca.dfn.de/pub/tools/net/mod_ssl/>`)

C.1 Browser-Relevantes

- Farrell McKay's **Fortify** - Starke Kryptographie für Netscape-Browser:
  - *http://www.fortify.net/*
    (*DFN-PCA Mirror* `<ftp://ftp.pca.dfn.de/pub/tools/net/Fortify/>`)
- **Opera** - Ein neuer Shareware-Browser mit starker Kryptographie:
  - *http://www.operasoftware.com/download.html*

C.2 Weitere Tools

- Netscape's **Certificate Server Extras**:
  - *http://developer.netscape.com/tech/security/certs/certs.html*
  - (*DFN-PCA Mirror für Solaris* **`<http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/ /solaris.cgi.tar.gz>`**)
- Microsoft's **CertMgr**:
  - *http://www.microsoft.com/security/tech/certificates/formats.asp#tools*
  - *http://msdn.microsoft.com/downloads/tools/authcodeie4/authcodeie4.asp?*
  - *http://www.microsoft.com/security/downloads/certinst.exe* (*DFN-PCA Mirror*)
- Peter Gutmann's **dumpasn1**:
  - *http://www.cs.auckland.ac.nz/~pgut001/dumpasn1.c*
  - *http://www.cs.auckland.ac.nz/~pgut001/dumpasn1.cfg*
- GMD's **SECUDE**:
  - *http://www.darmstadt.gmd.de/secude/*
- ...