Sicherheitsaspekte beim Apache Webserver

Seminararbeit zur Veranstaltung
Verteilte und parallele Systeme II
bei Prof. Dr. Rudolf Berrendorf

Von Andreas Fingerhuth und Timo Pick

Fachbereich Informatik
Fachhochschule Bonn-Rhein-Sieg, St. Augustin

1. Einleitung

Das World Wide Web hat sich in den letzten Jahren zu einem bedeutenden Kommunikationsmedium entwickelt, über das in zunehmendem Maße auch kommerzielle Transaktionen durchgeführt werden. Dies kann von Bestellungen mit Web-Formularen, über das Übertragen von Kreditkarteninformationen, bis hin zum Online-Banking gehen. Die Web-Server, von denen der Apache einen erheblichen Marktanteil besitzt, bilden die Grundlage für diese Kommunikation.
Das WWW in seiner heutigen Form bietet keine Datensicherheit.
Somit muss diese vom Web-Server zur Verfügung gestellt werden.

Insgesamt muss eine „sichere" Datenverbindung folgenden vier Punkten genügen:
Vertraulichkeit: Ein Spion kann aus den abgehörten Daten nicht den Inhalt ermitteln.
Integrität: Die übertragenen Daten können nicht unbemerkt verfälscht werden.
Authentizität: Die übertragenen Daten stammen nur vom richtigen Versender.
Unabstreitbarkeit: Der Sender einer Nachricht ist eindeutig identifizierbar.

Neben der Sicherstellung dieser vier Punkte ist ein dauerhafter, fehlerfreier und effizienter Betrieb des Servers wichtig. Diese Ziele können durch gezielte Angriffe von außen und innen in Gefahr geraten. Es muss sichergestellt werden, dass jeder Client nur die für ihn vorgesehenen Operationen ausführen kann und die Sicherheitsmechanismen nicht durch Hintertürchen ausgehebelt werden können. Aus diesem Grund führen wir in dieser Studie die Sicherheitsmaßnahmen, die im Apache-Server implementiert sind, auf. Des weiteren beschäftigen wir uns mit möglichen Angriffsszenarien und geeigneten Gegenmaßnahmen.

2. Angriffszenarien und Abwehr

Das Internet bietet keine expliziten Schutzmechanismen, um sich vor unerwünschten Zuhörern zu schützen. Auf jedem Router, über den die Kommunikation zwischen Client und Server läuft, kann ein Programm laufen, das diese **abhört** und speichert. Die so gespeicherten Kommunikationsinhalte enthalten Informationen über dass verwendete Protokoll und dass vom Client verwendete Passwort für die Anmeldung am Server. Generell kann ein Abhören durch Dritte nicht verhindert werden. Jedoch kann man die mitgehörte Kommunikation für den Zuhörer mittels Verschlüsselung unverständlich machen. Eine Variante, in der auch gesicherte Kommunikation abgehört werden kann, ist die **Man in the Middle** Attacke. Hiermit ist nicht nur das Abhören einer zwischen Client und Server geführten Kommunikation gemeint. Beim Verbindungsaufbau durch den Client gibt sich der Angreifer als Web-Server aus, an den die Kommunikationsanfrage gerichtet ist. Die Daten, die er so erhält, leitet er an den originalen Ziel-Server weiter und gibt sich bei diesem als der aus, von dem er die Daten erhalten hat. Durch diese Konstellation werden sämtliche Sicherheitsmechanismen umgangen. Der Angreifer erhält vom Client alle sicherheitsrelevanten Informationen. Erkennbar wird dieser Angriff lediglich dadurch, dass der Angreifer sich nur mit einem selbst erstellten Zertifikat legitimieren kann. Der Webbrowser des Clients macht den User darauf aufmerksam, dass das erhaltene Zertifikat nicht von einer als allgemein vertrauenswürdig anerkannten Institution ausgestellt wurde, setzt die Kommunikation nach Bestätigung durch den User aber fort. Ab diesem Zeitpunkt kann der Client, mit Ausnahme der etwas längeren Antwortzeit des Zielrechners, keinen Unterschied zur

Kommunikation ohne „Man in the Middle“ erkennen. Für den Angreifer sind alle Informationen sichtbar, die auch für den Client sichtbar sind.
Der Angreifer hat sich bisher darauf beschränkt, sich Daten, die nicht für ihn bestimmt waren, anzueignen. Bei der **Datenmanipulation** werden Daten abgefangen, gezielt verändert und dann weitergeleitet. Der Empfänger der Daten, der nichts von diesem Angriff weiß, denkt, dass der ursprüngliche Sender ihm diese Daten geschickt hätte. Im besten Fall hat der Angreifer die Daten so verändert, dass der Empfänger sie nicht mehr brauchen kann. Im schlimmsten Fall bleibt der Angriff unentdeckt. Die Datenmanipulation kann mit einem mitgelieferten Hashwert der abgesendeten Nachricht, die mit dem vom Empfänger selbst berechneten Hashwert verglichen wird, erkannt werden.
Eine wichtige Eigenschaft von öffentlichen Netzen ist, dass sie jederzeit zur Verfügung stehen und Dienste anbieten. Auf diesen Aspekt zielt der **(D)DoS-**Angriff. (D)DoS steht für (Distributed) Denial of Service. Ziel des Angreifers ist es hierbei, dass die Dienste z.B. des Web-Servers nicht mehr zur Verfügung stehen. DoS-Angriffe gehen von einem einzigen Rechner aus, DDoS-Angriffe von vielen Rechnern. Der Angreifer geht in beiden Fällen so vor, dass er an den Server so viele Anfragen stellt, dass dieser nur noch mit der Beantwortung dieser Anfragen beschäftigt ist, und Anfragen anderer Clients gar nicht mehr, oder nur sehr stark zeitverzögert, beantwortet. DoS-Angriffe können durch einen Paketfilter abgewehrt werden, indem dieser Pakete verwirft, wenn zu viele von einer einzigen IP-Adresse stammen. Bei DDoS-Angriffen geht man so vor, dass man über einen Application Level Gateway nur eine bestimmte Anzahl an Verbindungen zulässt.
Viren sind die Bedrohung von innen. Es sind nicht selbstständige Programmroutinen, die sich selbst reproduzieren und dadurch vom Anwender nicht kontrollierbare Manipulationen in Systembereichen, an anderen Programmen, oder deren Umgebung vornehmen [GSHB]. Man unterscheidet drei Arten von Viren:
Boot-Virus: Der Virus befindet sich im Bootsektor einer Diskette oder einer Festplatte und wird beim Warmstart, oder Kaltstart des Computers aktiviert.
File-Virus: Der Virus befindet sich in einer Datei und wird aktiviert sobald diese Datei aufgerufen wird.
Makro-Virus: Makros sind Steuerinformationen in Programmform. In diesen enthaltener schädlicher Code wird beim Aufruf des Makros gestartet.

3. Sicherheitsverfahren

3.1. Verschlüsselung

Unter Verschlüsselung versteht man die Ersetzung von Klartext unter Anwendung eines Algorithmus in Code, der nur lesbar ist, wenn man den Algorithmus und das "Shared Secret" kennt. Die Schwachstelle ist meistens nicht der Verschlüsselungsalgorithmen selber, sondern die Übermittlung des "Shared Secret".
Man unterscheidet symmetrische- und asymmetrische Verschlüsselung. Bei der symmetrischen Verschlüsselung wird mit dem gleichen Schlüssel entschlüsselt mit dem die Daten vorher verschlüsselt wurden. Bei der asymmetrischen Verschlüsselung gibt es ein Schlüsselpaar, bestehend aus zwei unterschiedlichen, aber zusammengehörenden Schlüsseln (öffentlicher und privater Schlüssel). Mit einer dieser Schlüssel wird verschlüsselt, mit dem anderen entschlüsselt. Mit welchem der beiden ver- oder entschlüsselt wird ist nicht bedeutsam.

3.2. Hash-Funktion

Die Hash-Funktion ermittelt aus den übermittelten Daten eine zwischen 128 (MD5) und 160 (SHA) Bit lange Zahl: Diese Zahl wird dann zusammen mit den Daten zum Empfänger geschickt. Der ermittelt wiederum den Hash-Wert der erhaltenen Daten und vergleicht ihn mit dem mitgeschickten Hash-Wert. Stimmen diese überein, so hat er denselben Text erhalten, den der Absender auch versandt hat.

3.3. Authentifikation

Die Authentifikation dient dazu, sicherzustellen, dass der Kommunikationspartner auch der ist, der er angibt zu sein. Grundsätzlich kann im Internet jeder behaupten er sei eine Person x. Beweisen, dass er es auch wirklich ist, kann er nur mittels Authentifikation. In der Regel erfolgt die Authentifikation über ein Passwort, welches nur die Peson x kennt. Hierzu muss vorher eine Identifikation mittels Username erfolgt sein. Eine allgemeinere Form der Authentifikation stellt die **Signatur** dar. Sie setzt sich aus zwei Komponenten zusammen.

1) das Zertifikat
2) Hash-Funktion

Das Zertifikat ist von einer Certification Authority ausgestellt und bestätigt die Identität des Inhabers dieses Zertifikates. Mit der angegebenen Hash-Funktion wird der Hash-Code des Zertifikates ermittelt und zusammen mit dem Zertifikat über die Public Key Infrastruktur an den Empfänger übertragen. Dieser kann die Daten mit dem öffentlichen Schlüssel des zu Identifizierenden wieder entschlüsseln, mit der Hash-Funktion die Datenintegrität prüfen und das Zertifikat prüfen, um den Absender sicher zu authentifizieren.
Das **Zertifikat** verlagert die Vertrauenswürdigkeitsfrage auf eine höhere Instanz, die bestätigt, dass der, der dieses Zertifikat besitzt, der ist, dessen Name auf dem Zertifikat steht. Hierfür steht die Certification Authority (CA) mit ihrem guten Namen in Form ihres Zertifikates gerade. Da sich jeder selbst Zertifikate ausstellen kann, ist es wichtig, dass man der CA vertraut. Es besteht eine CA-Hierarchie an deren Spitze in Deutschland das BSI steht.

3.4. Public Key Infrastruktur

Das Public Key Verfahren basiert auf asymmetrischer Verschlüsselung. Von einer zentralen Stelle wird jedem Nutzer ein Schlüsselpaar zur Verfügung gestellt, der private und der öffentliche Schlüssel. Dieses Schlüsselpaar wird genau einem Nutzer zugeordnet. Der öffentliche Schlüssel wird unter Angabe des Namens des Inhabers von der ausgebenden Firma in eine zentralen Directory öffentlich zur Verfügung gestellt. Der private Schlüssel ist nur dem Besitzer der Schlüsselpaares selber bekannt. Mit diesem kann er die von ihm ausgehende Kommunikation verschlüsseln. Der Empfänger der Nachricht beschafft sich aus einer Key Directory den öffentlichen Schlüssel des Senders und entschlüsselt mit diesem die erhaltene Nachricht. Da nur dem Sender der private Schlüssel bekannt ist, ist seine **Identität** sichergestellt.
Beschafft sich der Sender der Nachricht den öffentlichen Schlüssel des Empfängers und verschlüsselt mit diesem seine Nachricht, so kann nur der Empfänger mit seinem privaten Schlüssel die Nachricht entschlüsseln. In dieser Konstellation wird die **Vertraulichkeit** sichergestellt.

3.5. Firewalls

Firewalls sollen dabei behilflich sein, ungebetene Gäste vom Computer und somit vom Server fernzuhalten. Man unterscheidet im Wesentlichen zwei Arten. Mit **Paketfiltern** werden die ankommenden Datenpakete nach bestimmten Regeln ausgesiebt. So können Pakete, die von bestimmten IP Adressen stammen, generell verworfen werden, oder nur Pakete zugelassen werden, die über einen bestimmten sicheren Port, wie 80 (http) oder 25 (smtp), kommunizieren. Der Zugriff auf potentiell gefährliche Dienste, die auf anderen Ports laufen, wird so verwehrt. Ein **Application Level Gateway** (ALG) arbeitet auf der Anwendungsschicht und erlaubt nur bestimmten Diensten nach außen zu kommunizieren. Hierbei kommuniziert der Dienst zuerst mit einem durch den ALG erzeugten Proxy, der die Kommunikation dann an den eigentlichen Zielrechner weiterleitet.

3.6. Separate Netzwerke

Diese Implementierung ist ein Architekturmodell, was mit zwei Paketfiltern drei Zonen erzeugt, für die unterschiedliche Sicherheitsstandards aufweisen.

Zone 1:	Internet	ungesichert
Zone 2:	Demilitarisierte Zone	durch eine Firewall vom Internet getrennt
Zone 3:	innen	durch eine Firewall von der Demilitarisierten Zone getrennt

In Zone 2 steht ein nichtroutender Host, auf den von Zone 1und 2 zugegriffen werden kann.

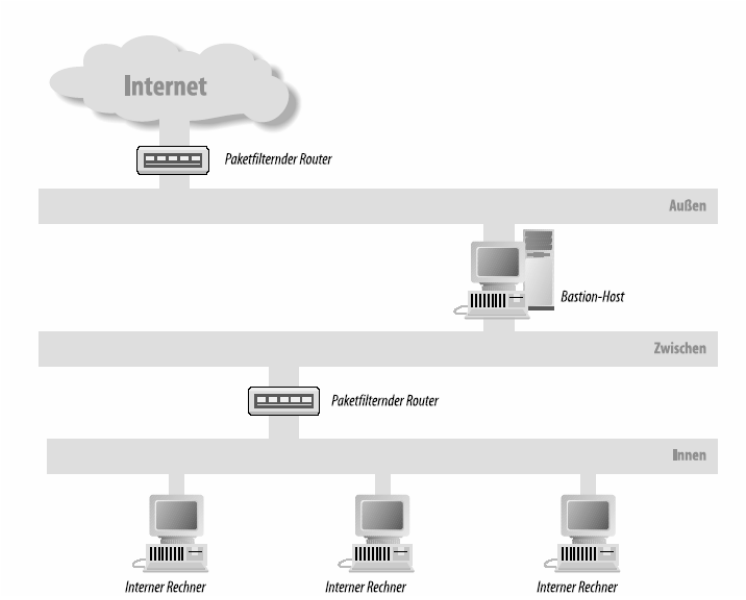

Abbildung 1: Separate Netzwerke

4. Grundlegende Sicherheitsbetrachtung Apache

Der Apache Web-Server wird stets mit Root-Identität gestartet. Dieser Prozess ist nur dafür zuständig, Verwaltungsarbeiten zu erledigen und Prozesse mit einem frei

einstellbaren Berechtigungsprofil zu erzeugen. Anfragen aus dem Internet beantworten nur die zweite Art von Prozessen. Dies hat den Vorteil, dass der Client keinen Kontakt zum „gefährlichen“ Vaterprozess mit Root-Rechten hat.

Das Common Gateway Interface (**CGI**) und Server Side Includes (**SSI**) bergen ein weiteres Gefahrenpotential. Man kann diese Dienste deaktivieren, was eine starke Einschränkung der Serverfunktionalität darstellt. Unter Linux ist dies relativ leicht mit dem Programm **suExec** zu lösen.
Ein weiterer Angriffspunk ist die Stärke des verwendeten **Zufallsalgorithmus,** der z.B. für den Aufbau einer SSL-Verbindung benötig wird. Nachdem man den eigentlichen Webserver sicher gemacht hat, muss man gewährleisten, dass es keine anderen Hintertüren gibt, um diese Sicherheit zu umgehen. Auf der Hardware, auf der der Webserver läuft, sollten für den Serverbetrieb entbehrliche Programme nicht betrieben werden. Je mehr Programme parallel ausgeführt werden, desto mehr Angriffspunkte findet ein Hacker vor.
Die aktuellen Sicherheitspatches für das Betriebssystem sollten aufgespielt sein, da nicht geschlossene Sicherheitslücken in Betriebssystemen gut dokumentiert und mit einer ausführlichen Einbruchsanleitung im Internet angeboten werden.

Allgemeine Verhaltensregeln:

Rechtevergabe:
Ein zentraler Punkt der Sicherheit ist die Zuteilung von Rechten. Diese bezieht sich nicht ausschließlich auf die Clients, die auf den Webserver regulär zugreifen. Ein erfolgreicher Angriff sollte stets in die Überlegung der Rechtezuteilung mit einbezogen werden. Grundsätzlich sollten nur die unbedingt notwendigen Rechte vergeben werden. Eine httpd.conf zu beschreiben zu dürfen ist z.B. nur für den root User von Bedeutung und sollte auch nur diesem gestattet werden. Allgemein kann man sagen, dass alle Programme und Dateien, die von Root User benutzt werden, auch nur durch diesen aufgerufen oder verändert werden können sollten. Daraus folgt direkt, dass sicherheitskritische Dateien (z.B. httpd.conf) nicht dem User gehören sollten, unter dessen ID der Server läuft, da bei einem Angriff dessen ID am schnellsten infiltriert ist. Aus demselben Grund sollten auch keine Veränderungsrechte oder Zugriffsrechte für ihn bestehen.

Zuordnung von Dateien
Aus Gründen der Administrierbarkeit sind getrennte Verzeichnisse für getrennte Dienste empfehlenswert, da nur so der Überblick über die angebotenen Dienste und deren Programme wie z.B. CGI-Skripte behalten werden kann. Bestimmte Verzeichnisse oder Dateien, die nicht zum öffentlichen Angebot des Servers gehören, sollten auch nicht im öffentlichen Bereich liegen. So muss man keine zusätzlichen Direktiven setzen, die den Zugriff auf solche Dateien explizit verbieten.

Um durch einen Servercrash oder ein Update die Zugriffsrechte wieder in alter Form herzustellen, sollten diese separat gespeichert und mittels Script nach einem Update wieder im Original herstellbar sein.

5. Apache Zugriffskontrollsystem

Ein Autor, der WWW-Dokumente verfasst möchte in der Regel auch, dass seine Dokumente von jedermann gelesen werden können. Deshalb gibt es in der Standard-Konfiguration von Apache keinen Zugriffsschutz, der das Lesen verbietet. Es gibt aber auch Inhalte, wie beispielsweise personenbezogene oder kostenpflichtige Daten, die nur bestimmten Benutzern oder Gruppen zugänglich sein sollten.

Das Stammverzeichnis der Webdokumente wird in der Direktive **`DocumentRoot /path/to/www`** festgelegt. Dies bedeutet aber nicht, dass nicht auch Webinhalte außerhalb des Stammverzeichnisses angefragt und beliefert werden können. Es gibt die Möglichkeit mittels symbolischer Links oder der Apache-Erweiterung **`mod_alias`** Dokumente, die woanders im System liegen, mit Apache zur Verfügung zu stellen.

Die XML-ähnlichen Direktiven **`<Directory />`** und **`</Directory>`** limitieren den Wirkungsbereich des eingefassten Abschnitts. In diesem Fall auf das UNIX-Wurzelverzeichnis `/` (läuft der Server unter Windows gilt dieselbe Direktive für den Windows-Arbeitsplatz, der alle Datenträger beinhaltet). Die Direktiven **`<Files datei.html>`** und **`<Limit PUT POST GET>`** ermöglichen eine Zugriffskontrolle auf einzelne Dateien und HTTP-Dienste.
Laut Maßnahmenkatalog des Grundschutzhandbuchs des BSI [GSHB M4.194] wird empfohlen das Wurzelverzeichnis mit folgender Regel gegen Zugriffe auf Dateien außerhalb der ***DocumentRoot*** abzusichern. Mit der Direktive ***Options*** kann angegeben werden, welche Serverfeatures im jeweiligen Verzeichnis erlaubt sind. Mit ***Options None*** werden Features wie Server-Side-Includes, CGI und das Ausführen symbolischer Links verboten. Motto: Es sollten nur jeweils die Optionen angewählt bleiben die gebraucht werden.

```
<Directory / >
Options None
AllowOverride None
Order Deny,Allow
Deny from all
</Directory>
```

Nachdem das Wurzelverzeichnis abgesichert ist, sorgt ein weiterer Limit-Abschnitt für die Freigabe der ***DocumentRoot***:

```
<Directory /var/www/htdocs >
Order Deny,Allow
Allow from All
</Directory>
```

Apache bietet ohne jede Erweiterung die Möglichkeit Webseiten nur einem bestimmten Publikum sichtbar zu machen. Diese Zugriffe können innerhalb des Servers wie folgt klassifiziert werden:

5.1. Zugriff von bestimmten Rechnern aus

Beim Apache-Webserver ist es sehr einfach möglich einzelne Rechner, Subnetze und Domains vom Zugriff auf bereitgestellte Seiten abzuhalten.

```
<Directory /var/www/htdocs >
Deny from .at .wir.fh-brs.de 195.125.73.22 192.168.0.
Allow from All
</Directory>
```

Order gibt die Reihenfolge der Auswertung der Regeln an. In diesem Beispiel werden zuerst die ***Deny***-Regeln, also alle Regeln die zu einer Verweigerung des Zugriffs führen können ausgewertet und danach die ***Allow***-Regeln. Das Schlüsselwort ***all*** bezeichnet alle Rechner im Netz. ***Host*** gibt die Rechnernamen an, auf die die Regel zutreffen soll. Einzelne Rechner und Rechnergruppen können anhand Ihrer IP-Nummer, ihrem vollständig qualifizierten Rechnernamen, oder anhand eines partiellen Domainnamen angegeben werden. In diesem Beispiel werden alle Rechner aus Österreich, alle Rechner des Fachbereichs Wirtschaft an der FH Bonn Rhein-Sieg, der Rechner mit der IP-Adresse 195.125.73.22 und alle Rechner des Klasse-C-Subnetzes 192.168.0.* vom Apache Server zurückgewiesen. Die ***Allow***-Regel ist eine so genannte Catch-All-Regel, wonach alle Rechner die nicht auf die obige Regel zutreffen, die Webseiten betreten dürfen.

5.2. Zugriff mit Abfrage von Benutzername und Passwort

Damit einem bestimmten Benutzer der Zugriff mittels Benutzername und Passwort der Zugriff auf einen geschützten Bereich gewährt werden kann, muss auf dem Serverrechner eine Passwortdatei angelegt werden. Dazu bringt die Apache-Suite den Befehl **`htpasswd`** mit.
In der Kommandozeile des Rechners oder mit einem Script kann man auf diese Art Benutzer-Passwort-Tupel in die Passwortdatei eintragen:

```
# htpasswd -cb /etc/apache/passwd TestUser strenggeheim
```

Das Passwort wird nach dem UNIX ***Crypt***-Verfahren des Betriebssystems gehashed. Wenn man Benutzergruppen einrichten will, reicht eine sehr einfache Zuweisung nach dem Schema
`Gruppe1: Benutzer1 Benutzer2 Benutzer3 Benutzer4…` in einer separaten Gruppendatei und die Direktive **`AuthGroupFile /pfad/zur/gruppendatei`** in der Konfigurationsdatei.

Will man einen passwortgeschützten Bereich innerhalb des Servers anlegen, geht das mit folgendem Abschnitt:

```
<Directory /var/www/htdocs/GEHEIM >
AuthName "Username + Passwort"
AuthType Basic
AuthUserFile /etc/apache/passwd
# AuthGroupFile /etc/apache/groups
Require user TestUser
# Require valid-user          // alternativ
```

```
# Require group Profs         // alternativ
</Directory>
```

Das Verzeichnis http://server/GEHEIMGEHEIM ist jetzt passwortgeschützt. **AuthType Basic** bedeutet dass eine Standardauthentisierung durchgeführt werden soll. Die Direktive **Require user TestUser** verlangt vom Besucher, dass er den Benutzernamen ***TestUser*** und das Passwort ***strenggeheim*** eingibt. Alternativ kann man mit **Require valid-user** einen beliebigen Benutzer aus der ***passwd***-Datei, oder mit **Require group Profs** einen Benutzer aus einer Gruppe ***Profs*** akzeptieren.

5.3. Benutzung bestimmter Dienste

In der Apache-Konfiguration lassen sich die HTTP-Dienste ***GET, POST*** und ***PUT*** auf dieselbe Weise wie Verzeichnisse schützen, oder Anfragen auf Seitenteile beschränken. Ist man sich beispielsweise sicher, dass ein Client keine HTTP-***PUT***-Anfrage stellen muss, so kann man dies auch von Apache verbieten lassen.

```
<Limit GET POST>
...
</Limit>
```

5.4. Kombination aus 1,2 und 3

Apache bietet die Möglichkeit die drei obigen Zugriffskontrollmechanismen zu verbinden. So können geschützte Ordner nur von bestimmten Benutzern mit Passwort von bestimmten Rechnern aus zugegriffen werden, oder ein öffentliches Verzeichnis gegen ***PUT*** und ***POST***-Anfragen geschützt werden.

```
Satisfy All
<Directory /var/www/htdocs/GEHEIM >
AuthName "Username + Passwort"
AuthType Basic
AuthUserFile /etc/apache/passwd
Require valid-user
Deny from .at .wir.fh-brs.de 195.125.73.22 192.168.0.
Allow from All
</Directory>
```

Mit der Direktive **Satisfy All** wird der Zugriff gewährt, wenn jede der Bedingungen (***valid-user*** und alle Rechner außerhalb der Beschränkungen) erfüllt. Anstatt ***All*** kann man mit dem Keyword ***Any*** akzeptieren, wenn nur eine der Bedingungen vom Client erfüllt wurde.

Dem Administrator steht es frei, die Seiten über die zentrale Konfigurationsdatei ***httpd.conf*** oder über einzelne ***.htaccess***-Dateien zu schützen. Die Direktiven in den Beispielen sind dieselben bis auf die <***Directory***>-Anweisung, die bei ***.htaccess*** entfallen kann. Gegebenenfalls überschreiben die Direktiven in der ***.htaccess***-Datei die der globalen Konfiguration. Mit der Direktive **AllowOverride None** kann dies

verhindert werden. Dadurch behält der Administrator die Autorität über den Zugriffsschutz der Webseite.

5.5. Die Standardauthentisierung *Basic:*

Bekommt der Apache-Webserver eine Anfrage auf eine Datei in einem geschützten Bereich so schickt dieser als Antwort einen ***`401 Authentication Required`*** Header zurück, der den Client auffordert, Benutzerkürzel und Passwort einzugeben. Da HTTP ein zustandsloses Protokoll ist, kommt diese Anfrage jedes Mal, wenn der Client auf eine geschützte Ressource zugreifen will. Allerdings merkt sich der Browser anhand der mitgeschickten ***AuthName*** in welchem geschützten Bereich er sich befindet, und übernimmt das Versenden von Username und Passwort bei der nächsten Gelegenheit selbsttätig. Die meisten Browser „merken“ sich die Verknüpfung zwischen ***AuthName*** und User-Passwort auch über den Neustart hinweg, wenn man die entsprechende Option angewählt hat.

Für die Authentisierungstypen ***Basic*** und ***Digest*** hält Apache verschiedene Erweiterungsmodule bereit:

- ***mod_auth_ldap*** und ***mod_auth_dbm***: Diese Module erlauben die Speicherung von Benutzernamen und Passwörtern für die Basic-Authentisierung auf einem LDAP-Server bzw. in einer Berkeley Datenbank, die Bestandteil der Berkeley Software Distribution (kurz BSD) und damit frei erhältlich ist.
- ***mod_access***: Dieses Modul realisiert die Zugriffssteuerung auf der Basis von IP-Adressen, Client-Hostnamen und anhand einiger anderer Kriterien.
- ***mod_auth_anon***: Dieses Modul erlaubt eine Zugriffsbeschränkung analog zum anonymen Zugriff auf FTP-Servern via HTTP-Basic. Dabei wird ein fest vorgegebener Benutzername abgefragt (etwa "anonymous") und der Besucher muss eine E-Mailadresse als Passwort angeben.

Die Apache Software Foundation rät aus einem einfachen Grund von der Verwendung der ***Basic***-Authentication bei relevanten Daten ab: Der Username und das Passwort werden unverschlüsselt übermittelt. Demnach gelänge es jedem Angreifer in demselben lokalen Netz die Benutzerdaten auszuhorchen und zu missbrauchen. Um dieses Manko zu beheben liegt dem Apache-Paket die Authentisierungsmethode ***mod_auth_digest*** bei. Dieses Modul implementiert das Digest-Authentisierungsverfahren, und wird durch die Direktive **`AuthType digest`** aktiviert.

Bei diesem Verfahren, dass im Prinzip dasselbe wie das ***Basic***-Verfahren ist, wird das Passwort nach der Eingabe durch den Benutzer mit dem UNIX-***Crypt***-Verfahren gehashed. Der Server vergleicht beide Hashes und gibt das Webdokument bei Übereinstimmung frei.

Bemerkungen :

- Da in der Stammkonfigurationsdatei ***httpd.conf*** der Ort der Passwortdatei und weitere sicherheitskritische Daten enthalten sind, muss sie vor inneren und äußeren Zugriffen unbedingt geschützt werden.
- Die einzelnen ***.htaccess***-Dateien dürfen nicht unbefugt gelesen werden, da sie ebenfalls den Ort der Passwortdatei enthalten. Deshalb sollte man den folgenden Abschnitt in die Stammkonfigurationsdatei eintragen, der Clientzugriffe verbietet:
  `<Files .htaccess>`

```
order allow,deny
deny from all
</Files>
```

- Das UNIX-***Crypt***-Verfahren, dass bei dem Hashen der Passwörter auf Server- und Clientseite verwendet wird ist schon relativ betagt und hat schwere Sicherheitsmängel. Für eine gesicherte Passwortübertragung empfiehlt die Apache Software Foundation den Einsatz von SSL über das Modul **`mod_ssl`**.

6. SSL/TLS-Unterstützung

HTTPS (=Hypertext Transfer Protocol Secure) ermöglicht eine durch Verschlüsselung gesicherte HTTP-Verbindung zwischen zwei Rechnern. Dabei werden die HTTP-Daten in verschlüsselte TCP-Pakete verpackt. Das bedeutet, dass nach dem Verbindungsaufbau der Client und der Server „abhörsicher" kommunizieren können. Die Stärke der Abhörsicherheit wird durch die eingesetzten Protokolle bestimmt. Zur Einrichtung einer SSL-verschlüsselten Verbindung wird ein asymmetrisches Verfahren und zum Übertragen der Nutzdaten ein symmetrisches Verfahren verwendet. SSLv3 und TLSv1 unterscheiden sich nur in sehr wenigen Punkten voneinander, weshalb die beiden Protokolle oftmals synonym behandelt werden.

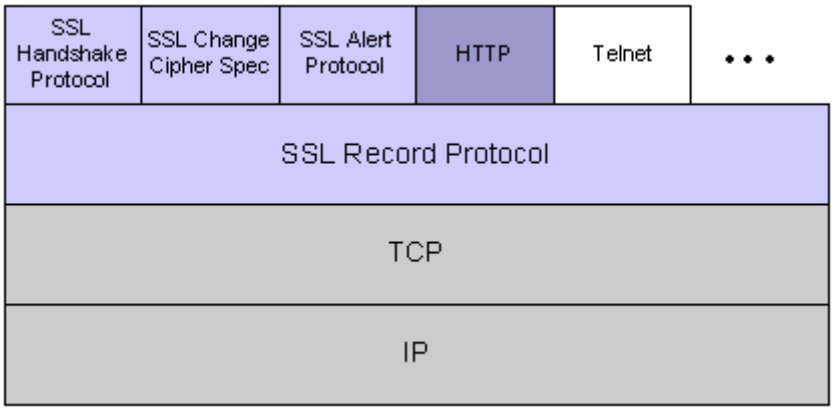

Abbildung 2: HTTP über SSL im Protokollstack

6.1. Historisches

„This module provides strong cryptography for the Apache (v1.3) webserver via the Secure Socket Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols by the help of the excellent SSL/TLS implementation library OpenSSL from Eric A. Young and Tim Hudson." [mod_ssl_13]

Es gibt verschiedene Implementierungen von SSL und TLS für den Apache Webserver. Die ursprüngliche Implementation namens ***Apache-SSL*** von Ben Laurie erschien zum ersten Mal in Form mehrerer Quellcodepatches und wurde gegen die ***SSLeay***-Libraries von Eric A. Young gebunden. Im April 1998 portierte der deutsche Entwickler Ralf S. Engelschall die Source-Patches der Version 1.17 für Apache 1.2.6 auf den Apache 1.3b6. Durch unterschiedliche Ansichten der beiden Entwickler entschied man sich, die Patches der Version 1.18 und die Sourcen des bis dahin unbedeutenden Moduls ***mod_ssl 1.x*** aufeinander abzustimmen. Dadurch entstand das Modul ***mod_ssl v2***, das im August 1998 veröffentlicht wurde. Nachdem im Jahr 2001 die USA ihre Exportbeschränkungen

für kryptographische Software gelockert haben, wurde ***mod_ssl*** in Apache 2 als Teil des Pakets integriert. Ab Apache 2 funktioniert die SSL-Unterstützung auch unter Microsoft Betriebssystemen. Mittlerweile hat sich ***mod_ssl*** so weiterentwickelt, dass nur noch ca. 5% des Sourcecodes von ***Apache-SSL*** und ***mod_ssl*** derselbe sind. Heutzutage wird das Modul gegen die ***OpenSSL***-Bibliotheken (Nachfolger von ***SSLeay***) gebunden. Neben den „offiziellen“ SSL-Implementierungen gibt es einige auf ***mod_ssl*** basierende kommerzielle Derivate (Raven SSL Module, Stronghold, RedHat Secure Web Server, etc.). Der Grund in der Verbreitung von kommerziellen SSL-Derivaten lag hauptsächlich darin, dass das Patent für den RSA Schlüsselaustausch bis ins Jahr 2000 von der RSA Security Solutions gehalten wurde und man teure Lizenzen für dessen Verwendung kaufen musste. Da das Patent seit September 2000 abgelaufen ist, rückten die offenen Implementationen von SSL immer mehr in den Vordergrund und sind heutzutage marktbeherrschend.

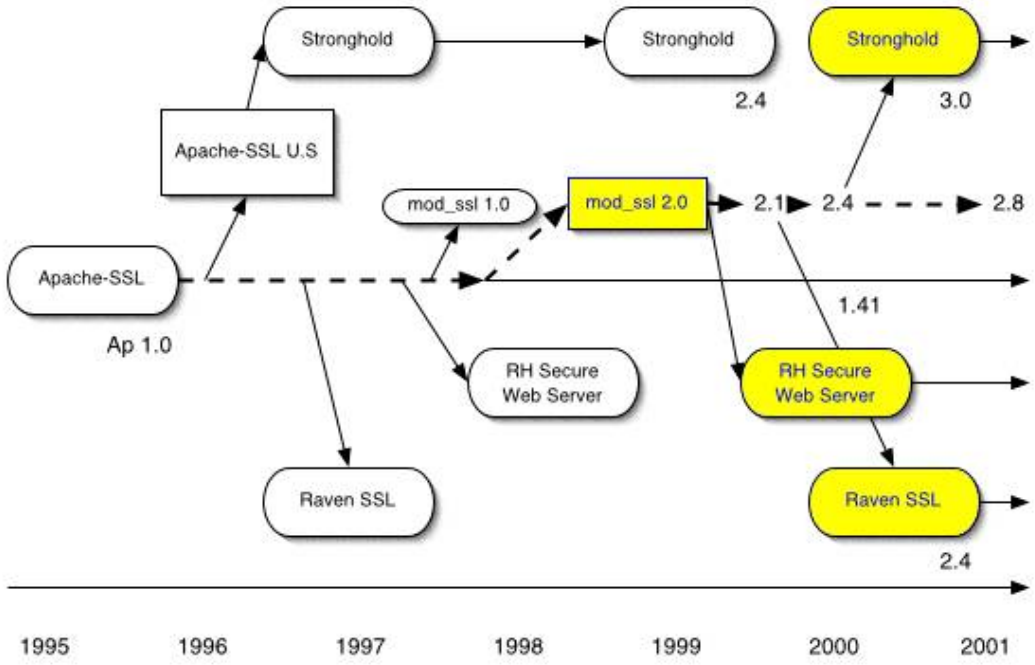

Abbildung 3: Apache + SSL Timeline

6.2. Apache mod_ssl

Das mod_ssl-Modul von Apache wird gegen die OpenSSL-Bibliotheken gebunden. OpenSSL stellt die konkreten Implementierungen der Verschlüsselungsverfahren und eine umfangreiche API zur Verfügung, womit darüberliegende Protokolle leicht um die SSL-Verschlüsselung erweitert werden können. Zur Realisierung von HTTPS wird OpenSSL als ***Filter*** verwendet – das bedeutet, dass Clientkommandos am Port 443 von der Apache Serverinstanz entgegengenommen, an OpenSSL zur Entschlüsselung durchgereicht und danach wieder an Apache hochgegeben werden. Der Server antwortet indem er das HTTP-Antwortpaket an OpenSSL übergibt, die verschlüsselten Daten in TCP-Segmente verpackt und an den Client schickt. Für die Einbindung von OpenSSL in Apache 1.3 (***Abbildung 3***) bedurfte es zum einen dem Modul mod_ssl (Punkt 1) als Adapter für die OpenSSL-Bibliotheken, und zum andern einer erweiterten Apache API (Punkt 2).

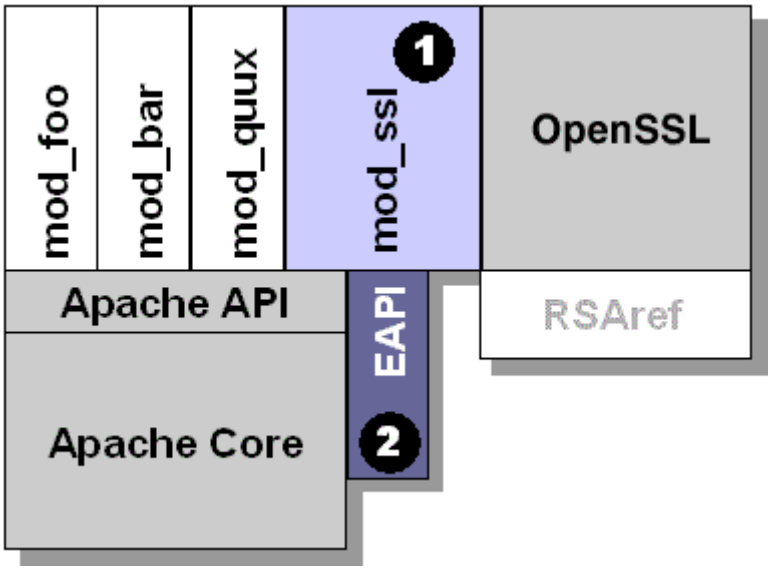

Abbildung 4: Apache 1.3, mod_ssl und OpenSSL

Im Folgenden wird nur auf die Built-In-Implementierung ***mod_ssl*** von Apache 2 eingegangen. Die neue Architektur des Apache2-Webservers erlaubt es, die SSL-Unterstützung bereits bei der Kompilierung zu aktivieren, und das Module dynamisch ge- und entladen (***DSO***) werden können. Die Verwendung von ***DSOs*** (=Dynamic Shared Objects) machte die Apache 1.3 EAPI überflüssig. Bei der Kompilierung von Apache2 mit mod_ssl müssen lediglich die Switches **`--enable-ssl --with-path=/path/to/OpenSSL/lib`** gesetzt sein. Mit dem in der Apache Suite enthaltenen Apache Extension Tool ***apxs*** können Erweiterungsmodule auch noch nachträglich kompiliert und eingebunden werden. Nach der Kompilierung aktiviert man in ***httpd.conf*** an den gewünschten Stellen (typischerweise serverweit oder in VirtualHost-Blöcken) wie im folgenden Beispiel die SSL-Unterstützung:

```
Listen 443
SSLSessionCache /pfad/zu/ssl/gcache
SSLMutex sem
<VirtualHost _default_:443>
ServerName geheim.gemein.de
SSLEngine on
SSLCertificateFile /pfad/zu/Zertifikat.cert
SSLCertificateKeyFile /pfad/zu/Zertifikat.key
</VirtualHost>
```

Der ***well-known*** Port für HTTPS ist 443, deshalb sollte der Server auf diesem Port horchen. SSL arbeitet mit einem separaten Session-Key für jede Verbindung. Beim Aufbau der Verbindung werden Zertifikate geprüft und der Session-Key vereinbart. Da dies eine zeitaufwändige Angelegenheit ist, ist es möglich diese Session-Keys wieder zu verwenden, wenn derselbe Client erneut anfragt. Da Apache aber mit vielen Prozessen oder Threads (je nachdem auf welchem Betriebssystem es läuft) die Anfragen der Clients beantwortet, gibt es einen globalen Session-Cache (gcache), der alle wichtigen Informationen für die Wiedererkennung der Clients speichert und so den Verbindungsaufbau beschleunigt**.** Den Ort gibt des gemeinsamen Caches gibt ***SSLSessionCache*** an, die Art des Multi-User-Lock wird mit ***SSLMutex*** gesetzt. Die Standardeinstellung von ***SSLMutex*** ist ***none***, was man unbedingt ändern sollte, es sei denn man nimmt einen zuweilen durcheinander gebrachten Session-Cache in Kauf. Eine

sinnvolle Einstellung kann **`SSLMutex sem`** sein, das automatisch die beste vom Betriebssystem bereitgestellte Methode (bei Unix pthread) wählt. Ebenso ist es möglich den Session-Cache in einer Datenbank verwalten zu lassen. Die Direktive, die für einen Bereich SSL aktiviert ist **`SSLEngine on`** . Das Serverzertifikat, bestehend aus öffentlichem Schlüssel und Signatur einer CA, sowie die dazugehörige Passphrase werden mit ***SSLCertificateFile*** und ***SSLCertificateKeyFile*** gesetzt. Diese Daten sind unbedingt vor unbefugtem Zugriff zu schützen. Wer sicher gehen möchte der legt die Passphrase nicht auf dem Rechner ab, sondern gibt sie bei Serverstart von Hand ein.

6.3. Zertifikat erstellen

Hat man Apache und mod_ssl konfiguriert, muss ein Serverzertifikat erstellt werden. Für den Testbetrieb stellt Apache mit dem Kommandozeilenbefehl **`make certificate`** ein Testzertifikat zur Verfügung. Vorsicht: Dieses Zertifikat ist selbstsigniert. Doch damit ein Client sicher sein kann, dass es sich tatsächlich um den gewünschten Server handelt muss das Zertifikat von einer Certification Authority (CA) signiert werden. Die meisten Browser werden eine Warnung anzeigen, wenn sie die Seite betreten. Ein signiertes Zertifikat kann man nur dann erhalten, wenn die Personalien einer höheren Instanz bekannt gemacht und verifiziert wurden. Viele CAs lassen sich für das Ausstellen einer Signatur bezahlen. Einen kostenlosen Weg ein signiertes Serverzertifikat zu erhalten stellt www.cacert.org bereit.

Die größte Schwachstelle des HTTPS-Protokolls ist die Abhängigkeit von signierten Zertifikaten. Ohne ein fremdsigniertes Zertifikat ist das Protokoll verwundbar gegenüber dem "man in the middle"-Angriff. Dabei gibt sich ein Angreiferrechner als der gewünschte Rechner aus und empfängt und beantwortet die Clientanfragen. Ist das Zertifikat des wirklichen Rechners unsigniert reicht es, dass der Angreifer das Verhalten des echten Rechners genau nachahmt. Der Client wird eine Warnung erhalten, der Rechner habe ein selbstsigniertes Zertifikat, da er aber diese Warnung vom echten Rechner kennt wird er sie ignorieren.

6.4. Cipher-Suites

Eine Cipher-Suite stellt keine eigenständige Sicherheitsvorrichtung dar, sondern kombiniert verschiedene Sicherheitsmechanismen. Sie erleichtert die Bedienbarkeit der Software und kombiniert Verfahren mit der Absicht hochperformanter Kommunikation, hochsichere Kommunikation, oder ein Mix aus beidem, zu unterstützen.

Eine Cipher-Suite ist ein 5-Tupel bestehend aus SSL/TLS-Version, Schlüsselaustausch-verfahren, symmetrischem Verfahren, Schlüssellänge und Hashingverfahren, das eine Sicherheitsumgebung spezifiziert. ***mod_ssl*** (bzw. OpenSSL) unterstützt unter anderem folgende Verfahren:

Symmetrische Verfahren:
3DES
RC4
RC2
DES
IDEA
AES

Asymmetrische Verfahren:
RSA
Diffie Hellman

Hashingverfahren:
MD5
SHA-1

mod_ssl unterstützt die Versionen SSL v.1, SSL v.2, SSL v.3 und TLS v.1. Die vom Server akzeptierten Versionen können in der Stammkonfigurationsdatei ***httpd.conf*** über die Direktiven ***SSLProtocol*** und ***SSLCipherSuite*** an- und abgewählt werden. Für die meisten Verwendungszwecke ist es ratsam die SSL-Protokollversionen SSLv1, SSLv2 und schwache Verschlüsselungsverfahren (vor allem der ***„No-Encryption-Modus“***) auszuschließen.

```
SSLProtocol ALL –SSLv1 –SSLv2
SSLCipherSuite ALL:!NULL
```

Dies kann allerdings zur folge haben, dass ältere Browser ohne ***SSLv3*** oder ***TLS*** - Unterstützung die Webseite nicht anzeigen können. Um dies zu verhindern kann man die starke Verschlüsselung auf die wirklich kritischen Teile der Webseite begrenzen. Für einfache Handhabung der verschiedenen Suites gibt es die Aliase ***LOW***, ***MEDIUM*** und ***HIGH.***

```
# Erlaube einfache Cipher und Verschlüsselung
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

# aber https://hostname/streng/geheim und
# tiefer verlangt starke Protokolle
<Location /streng/geheim>
SSLCipherSuite HIGH:MEDIUM
</Location>
```

6.5. Benutzerauthentisierung via HTTPS

Neben den Server-Zertifikaten können auch signierte Client-Zertifikate nach ***ITU-T X.509.3*** erstellt werden. Dies ermöglicht eine Authentifizierung der Clients gegenüber dem Server und gilt als die momentan sicherste Methode der Benutzerauthentisierung im WWW. Moderne Browser können eigene Zertifikate verwalten und beim Eintritt in eine HTTPS-Verbindung die Authentisierung vornehmen, ohne das der Benutzer angehalten wird ein Passwort einzugeben. Dazu benötigt man ein Clientzertifikat, das von derselben ***CA*** signiert ist wie das Serverzertifikat oder im Rahmen des PKI-Durchlaufs vom Server ermittelt werden kann. Oftmals werden Clientzertifikate mit einer Lebensdauer (***expiration date***) versehen, die verhindert dass ein einmal signiertes Clientzertifikat ewig gültig ist. Zudem gibt es die Möglichkeit auf Serverseite eine ***CRL (Certificate Revocation List)*** zu führen, in der gültige aber unerwünschte Zertifikate aufgelistet werden. So könnte es passieren, dass ein Clientzertifikat in die falschen Hände fällt. Clients, deren Zertifikate in dieser Liste stehen erhalten keinen Zugriff auf den geschützten Bereich. Auch dieser Sicherheitsmechanismus benötigt ein bisschen Konfigurationsarbeit. Die SSL-Benutzerauthentisierung lässt sich ähnlich granular wie die Basic-Authentisierung in Directory oder VirtualHost-Blöcken einsetzen. Eine einfache und weit verbreitete Methode die Benutzer und Gruppen mit Clientzertifikaten zu verwalten ist mit `FakeBasicAuth` gegeben. Der Benutzername in der Passwortdatei wird aus dem ***Subject Distignuished Name (DN)*** des Zertifikats abgeleitet, das nicht verwendete Passwort wird auf einen Standardwert gesetzt. Danach kann man die Mechanismen und Direktiven der HTTP ***Basic*** Autorisierung verwenden. Damit die Clientzertifikate als gültig erkannt werden, muss das Zertifikat der ***CA*** importiert werden.

Mit **`SSLVerifyClient require`** weist man den Server an, nur Clients mit gültigem Zertifikat zuzulassen. ***SSLVerifyDepth*** gibt die Verifizierungsstiefe an. Der Server wandert um die dort spezifizierte Anzahl in der Verifizierungshierarchie hinauf um den CA zu finden, der das Clientzertifikat signiert hat. Mit **`SSLVerifyDepth 1`** werden nur Clients akzeptiert, dessen CA direkt dem Server bekannt ist.

```
<VirtualHost _default_:443>
SSLVerifyClient require
SSLVerifyDepth 0
SSLCACertificateFile /pfad/zu/CAZert.cert
SSLOptions FakeBasicAuth
AuthUserFile /etc/apache/passwd
Require valid-user
</VirtualHost>
```

Um CRL zu aktivieren erstellt man ein CRL-Verzeichnis und kopiert die Datei mit den unerwünschten Zertifikaten (hier: revoked.crl) und setzt innerhalb des Directory oder VirtualHost-Blocks folgende Direktiven:

```
SSLCARevocationPath /pfad/zu/crl/
SSLCARevocationFile /pfad/zu/crl/revoked.crl
```

Damit ein Client tatsächlich von Apache zurückgewiesen wird muss man im CRL-Verzeichnis einen symbolischen Link setzen, dessen Dateiname der Hashwert des zurückgewiesenen Zertifikats ist.

```
% ln -s ca.crl `openssl crl -hash -noout -in revoked.crl`.r0
```

Zum Schluss der Betrachtungen soll noch einmal ein fast komplettes Beispiel veranschaulichen, wie Zugriffsschutz mit den beschriebenen Mechanismen ineinander greifen kann:

```
<VirtualHost 192.168.0.36:443>
       SSLCertificateFile /www/conf/ssl.crt/mysite1.crt
       SSLCertificateKeyFile /www/conf/ssl.key/mysite1.key
       SSLEngine on
       DocumentRoot /www/virtualhosts/mysite1
       ServerName mysite1.mycorp.com
       SSLCACertificatePath /www/conf/ssl.crt
       <Directory "/www/virtualhosts/mysite1/">
              SSLVerifyClient require
              SSLVerifyDepth 10
              SSLOptions FakeBasicAuth
              AuthUserFile /etc/apache/passwd
              Require valid-user
       </Directory>
       <Location "/">
              AuthName "mysite1 login"
              AuthType Basic
              Auth LDAPHosts "ldap.mycorp.com:389"
              AuthLDAPBaseDN "dc=mycorp,dc=com"
              AuthLDAPUserKey uid
              AuthLDAPPassKey userpassword
              AuthLDAPGroupKey webgroups
              AuthLDAPSearchScope subtree
              <Limit GET POST>
                     require group mysite1
              </Limit>
```

```
      </Location>
</VirtualHost>
```

7. Logfiles des Servers

Logfiles sind von entscheidender Bedeutung für den Betrieb eines Webservers. Mit Ihrer Hilfe können sowohl generelle Serverprobleme, wie auch Angriffe auf den Server erkannt werden. Werden Pfade für Protokolldateien nicht explizit angegeben, so werden diese im Unterverzeichnis *logs* des Installationsverzeichnisses abgelegt. Da Protokolldateien sehr schnell eine beachtliche Größe erreichen können, droht ein Überlaufen der entsprechenden Partition. Dies kann zu ernsthaften Störungen des Serverbetriebs führen [GSHB].
Der Apache benutzt eine Logginghierarchie um zu entscheiden, wie ausführlich in seine Logfile geschrieben wird. Insgesamt stehen acht Hierarchiestufen zur Verfügung. Hierbei ist darauf zu achten, ein ausgewogenes Gleichgewicht zwischen ausführlichem Informationsinteresse des Sicherheitsbeauftragten und der Größe der Logdatei zu finden. Die Online-Dokumentation des Apache-Webservers empfiehlt, mindestens den LogLevel *crit* zu benutzen, so dass alle Emergencies, Alerts und Critica Conditions in das ErrorLog geschrieben werden [GSHB]. Insgesamt bietet der Apache folgende Loglevel (geordnet von minimalen hin zum ausführlichen Logen):

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

Neben der einzigen im Kern realisierten Möglichkeit, Logdateien zu schreiben (ErrorLog), kann das Modul mod_log_config eingebunden werden, um weitere Logdateien zu ermöglichen.
Es können auch Anfragen, die über das http Protokoll erfolgen, protokolliert werden. Diese werden dann in die AccessLog geschrieben. Folgende Inhalte können in die Logdatei geschrieben werden.

- 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.

Per default werden Inhalte im Common Log Format in das Logfile geschrieben. Hierbei werden der **Hostname des Clients**, der **Name des Benutzers** auf dem Client (sofern dieser durch den identd Dienst auf dem Client identifizierbar ist),der **Benutzername als Ergebnis des Authentisierungsprozesses**, die **Zeit des Zugriffs**, die **erste Zeile des**

HTTP-Requests, der **Status des HTTP-Requests** und die **Anzahl der übertragenen Bytes** protokolliert.
Mit der Direktive **`HostnameLookups`** kann bestimmt werden, ob die IP Adresse (**`off`**), der symbolische Namen (**`on`**), oder das Ergebnis des DNS Reverse Lookup nach einer DNS Überprüfung, die aktuelle IP Adresse des Clients liefert (**`double`**), protokolliert wird. Speziell die letzte Einstellung verursacht starke Performanceverluste und ist daher nur bei Hochsicherheitssystemen zu empfehlen.
Das Modul *mod_usertrack* gestattet mit Hilfe von Cookies die Verfolgung der "Spur" eines Benutzers durch einen Webserver. Aus ihr lässt sich erkennen, welche Seiten in welcher Reihenfolge und in welchem Zeitabstand von einem Benutzer angefordert wurden.

Inhalte der Logfiles
Überwacht werden sollten folgende Punkt:

- Richtlinienänderungen
- Anmeldeereignisse
- Anmeldeversuche
- Kontenverwaltung

Sicherheitsaspekte beim Loggen
Die Logdatei wird stets vom obersten Webserverprozess geöffnet und der Filehandler dann an die schreibenden Prozesse übergeben. Damit erfolgt die Eintragung immer unter Rootrechten. Lokale Benutzer dürfen keine Schreibrechte auf diese Datei haben, damit sie sie nicht manipulieren, oder durch eine manipulierte Datei ersetzen können.
Generell reicht es aus, wenn nur Root auf die Logdateien zugreifen kann. Sollten andere User ein berechtigtes Interesse am Lesen von Logdateien haben, so kann dies auf zwei Weisen erfolgen.
1) Über einen in mod_log_config bereitgestellten Mechanismus zum bedingten Logging werden verschiedene Logdateien angelegt, in die jeweils nur die Zugriffsdaten der einzelnen Benutzer bzw. virtuellen Hosts protokolliert werden.
2) Es wird in eine zentrale Datei geloggt. Später wird der Inhalt dieser Datei mit einem Programm in Logdateien verteilt, für die User Zugriffsrechte besitzen.
Die Apache-Distribution enthält bereits ein Programm (*split-log*), das zur Verteilung der Protokollierungsdaten auf mehrere Logdateien eingesetzt werden kann.
Die zweite Variante ist vorzuziehen, da weniger Serverlast erzeugt wird (weniger offene Logfiles) und eine zentrale, geschützte Logdatei zur Auswertung zur Verfügung steht.
Archivierung
Bei der Archivierung von Logdateien ist darauf hinzuweisen, dass der Apache Server, so lange er läuft, die Logdatei geöffnet hält und in sie hineinschreibt. Da eine Archivierung von geöffneten Dateien nicht möglich ist, muss der Server angehalten werden, um die Logdateien zu speichern. Sollte es sich um einen kommerziellen Server handeln, ist dies kein gangbarer Weg. Apache bietet die Möglichkeit, Protokolldaten über eine Pipe an ein externes Kommando zu übergeben, dass sich um das „Rotieren“ von Protokolldateien kümmert. Das in der Apache-Distribution mitgelieferte Programm *rotatelogs* bietet die Möglichkeit nach festgesetzten Zeitabschnitten eine Logdatei zu schließen und in eine neu zu schreiben. Der Befehl :

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

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

Auswertung
Das Schreiben von Logdateien alleine bringt noch keinen Sicherheitszuwachs. Erst das Auswerten fördert Angriffsversuche zu Tage. Da Logdateien oft sehr groß und unübersichtlich sind, gibt es spezielle Programme, die einen dabei unterstützen.
Die ErrorLog ist für die Systemsicherheit essentiell, da sie Fehlercodes enthält, die unter anderem durch Angriffsversuchen entstehen. Das Tool *Analog* bietet einen *Failure Report* an, in dem es fehlgeschlagene HTTP-Requests zusammenfasst. Diese entstehen oft bei Angriffen auf Webserver. Indiz für einen Angriff kann sein:

- eine Häufung von Anfragen von einer bestimmten IP-Adresse
- eine Häufung von Anfragen nach einer bestimmten Datei
- eine Häufung des HTTP-Fehlers 404 - Datei nicht gefunden
- eine Häufung des HTTP-Fehlers 403 - Zugriff verweigert
- Anfrage nach CGI Skripten, die als fehlerhaft bekannt sind
- URLs, die viele Zeichen enthalten, die z. B. in der Form %20 kodiert sind
- wiederholte Anforderungen von auf dem Webserver nicht vorhandenen Seiten
- überlange URLs

Eine vollständige Anleitung zum Erkennen von Angriffen durch die Auswertung von Logdateien würde den Rahmen dieser Ausarbeitung sprengen. Zu erwähnen ist aber noch, dass eine Sicherungskopie der Logdateien auf einem anderen Rechner in regelmäßigen Abständen das Risiko des Verlustes oder der nachträglichen Manipulation solcher Dateien minimiert.

8. Einfluss des Betriebssystems

Unabhängig davon, welches Betriessystem installiert ist, können Probleme durch zur Verfügung gestellten **Ressourcen** auftreten. Z.B. kann es durch eine zu kleine Partition für den Apache zu Betriebsstörungen kommen. Des weiteren müssen die Standard-einstellungen kritisch überprüft werden. Besonderes Augenmerk ist auf die Anzahl der unterschiedlichen User und deren Berechtigungen zu legen. Bei einem Webserver sind zwei Arten von User zu unterscheiden. 1) Diejenigen, die aus dem Internet heraus zugreifen, „externe“ User genant und 2) die, für die Administration und Aktualisierung des Servers zuständig sind, die „internen“ User. Hier gilt die Regel: Alles verbieten und dann nur das unbedingt Notwendige erlauben. Die Zugriffsrechte auf die Verzeichnisse des Apache Webservers können durch ein eigenes Account auf dem Betriebssystem realisiert werden. Auch hier gilt die oben erwähnte Regel.
Wie den Apache Server selbst, ist es auch hier wichtig, das Betriebssystem schlank zu halten. Dies wird, durch das Deaktivieren oder besser noch das Deinstallieren nicht benötigter Dienste erreicht. Außerdem können Dienste wie Telnet- oder ein ftp Client, den Angreifer unterstützen weitere aggressive Programme von seinem Rechner auf den Zielrechner zu übertragen.
Im Gegensatz zu seiner Vorgängerversion ist der Apache 2.0 betriebssystem-unabhängig. Lediglich das Modul *Apache Portable Runtime* (APR), worauf die anderen Apache-Module aufsetzen, ist betriessystemabhängig. Es bietet die grundlegenden Funktionen eines Betriebssystems, wie die Ein- und Ausgabe von Dateien, Netzwerkfunktionalität, Thread- und Prozessverwaltung, Speicherverwaltung und das Laden dynamischen Codes.

8.1. UNIX / Linux

Ein Großteil der Sicherheit von Linux beruht auf seinem Dateisystem. Nahezu jede Systemressource wird als Datei abgebildet und besitzt die bekannte Berechtigungsmaske:

```
-rwxr-x--- root root Dateiname
```

Ist das x-Bit für den entsprechenden Benutzer gesetzt, darf das Programm unter der Benutzer-ID des Aufrufers ausgeführt werden. Diese Rechte können durch die Befehle **`chmod`** oder **`chown`** modifiziert werden.
Bei gesetztem S-Bit wird das Programm entweder mit der ID des Besitzers oder der ID der Gruppe gestartet. Dies hat negative Sicherheitsauswirkungen. In diesem Beispiel:

```
-rwsr-xr-x root root Programm1
```

führt jeder User dieses Programm mit Rootrechten aus. Mit dem Befehl:

```
find / -type f \( -perm +4000 -o -perm +2000 \) –print
```

werden alle Programme mit gesetztem S-Bit aufgelistet. Diese sollte man auf Notwendigkeit des S-Bits kontrollieren.
Neben der sinnvollen Vergabe solcher Rechte ist eine sichere Prüfung der Authentität wichtig. Der Standardalgorithmus von Linux ist hier nicht die beste Wahl. Es ist empfehlenswert diesen durch das ***Pluggable Authentication Module*** (PAM) zu ersetzen. Dies ist z.B. bei SuSE-Linux ab der Version 6.2 bereits integriert. Eine Nachrüstung ist schwierig, da sämtliche Dienste, die Authentifizierung verwenden, neu kompiliert werden müssen.
Möchte sich ein Benutzer bei einem Dienst authentifizieren, so leitet dieser die Anfragen an PAM weiter und erhält nach Prüfung durch PAM eine Freigabe oder Ablehnung für den Benutzer.

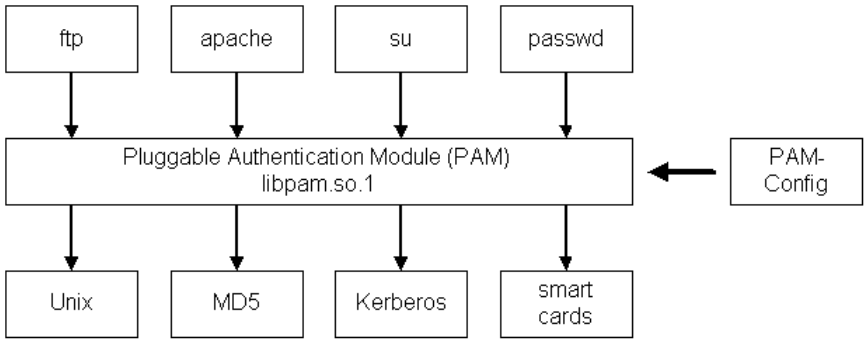

Abbildung 5: PAM

Unter Linux wird Apache mit einem Prozess mit Rootrechten gestartet. Dieser Prozess übernimmt nur Verwaltungsaufgaben. Die eigentlichen Webanfragen beantworten von diesem Prozess gestartete Kinderprozesse, die mit eingeschränkten Rechten ausgestattet sind. Für jede Anfrage wird ein Prozess gestartet.

Wie eben schon erwähnt, stellen Zugriffsrechte hier einen wesentlichen Aspekt für die Sicherheit dar.
Mit dem ***inetd***-Dienst, der unter anderem dafür sorgt , dass nur selten benötigte Daemons nicht ständig im Hintergrund laufen, sondern bei Bedarf gestartet werden, kann über den Umweg des ***TCP-Wrapper*** eine weitere Zugriffskontrolle erfolgt. Dazu muss bei der Anfrage zum Starten eines Dienstes der Serverpfad **`/user/sbin/tcpd`** gesetzt sein. Der TCP-Wrapper prüft unter Verwendung der Dateien

`/etc/hosts.allow` und **`/etc/hosts.deny`** (auch in dieser Reihenfolge) die Zugriffsberechtigung. Hierbei gilt, dass in der hosts.deny nichts verboten werden kann, was in der hosts.allow erlaubt wurde. Leider weist dieser Mechanismus eine Schwachstelle auf. Wird in beiden Dateien kein Eintrag gefunden, so wird der Dienst trotzdem gestartet.

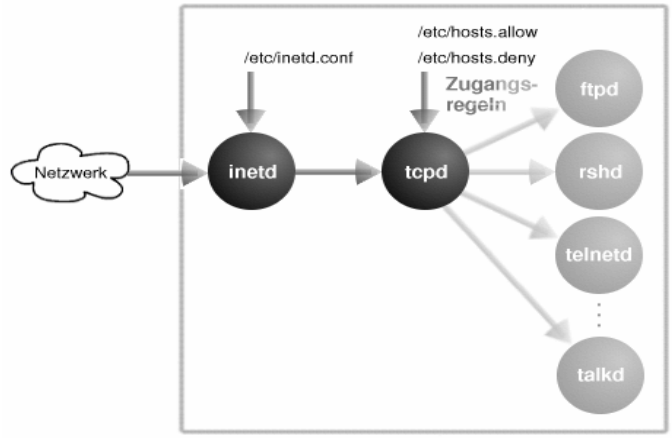

Abbildung 6: TCP-Wrapper

Die Konfigurationsdatei **`/etc/inetd.conf`** enthält eine Liste der aktiven Dienste, die nicht zum Systemkern gehören und von ***inetd*** unter der oben aufgeführten Einschränkung aufgerufen werden können. Um Änderungen in der **`inetd.conf`** zu übernehmen, muss das Kommando

```
killall -HUP inetd
```

ausgeführt werden.

Bei der Verwendung von CGI und SSI stellt das Programm **suExec** eine Hilfe dar. Mit dessen Unterstützung kann die User-ID für CGI und SSI gesetzt werden. Voreingestellt ist die ID des Servers, der diesen Dienst ausführt. Hat man dessen Rechte stark eingeschränkt so ist eine weitere Einschränkung mit **suExec** nicht notwendig. Leider steht ein solches Programm unter Windows nicht zur Verfügung. Hier ist man darauf angewiesen die ankommenden Daten auf potentiell gefährliche Inhalte zu untersuchen.

Der Webserver sollte zusätzlich in einer Chroot-Umgebung betrieben werden. Diese bewirkt, dass die Prozesse von Apache nicht auf Dienste, Verzeichnisse oder Dateien außerhalb dieser Umgebung zugreifen können. Damit ist es dann auch nicht mehr möglich, Systemdateien des Betriebssystems aufzurufen. Vom Server benötigte Ressourcen, die außerhalb dieser Umgebung liegen, sind ebenfalls nicht mehr zugänglich und müssen in diese Umgebung kopiert werden.

8.2. Windows

Generell gilt es unter Windows Augenmerk auf dieselben Punkte zu legen, wie bei Linux. So erlauben die Default-Einstellungen z.B. allen Usern einen Vollzugriff auf die Logdateien, was zu verbieten ist. Auch das verwendete Dateisystem ist bedeutsam. Während *FAT* nur grobgranulare Rechtevergaben und Benutzertrennung unterstütz, ist *NTFS* hier wesentlich feingranularer. *EFS* (Encrypted File System) basiert auf NTFS und stellt eine verschlüsselte Ablage der Daten sicher. Daten werden so vor Angreifern die bereits Zugriff auf die Hardware haben geschützt. Zugreifbar bleiben diese Dateien jedoch, wenn nicht unter dem originalen Betriebssystem gestartet wird. Abhilfe schafft

hier ein SYSKEY-Kennwort, was den Systemstart nur nach Eingabe dieses Kennwortes erlaubt. SYSKEY bietet drei Varianten, wie das Kennwort einzugeben ist.

1) Abfrage durch das Betriebssystem und Eingabe über die Tastatur.
2) Das Kennwort auf dem Computer selber gespeichert und automatisch geladen.
3) Das Kennwort wird auf Diskette gespeichert, die beim Systemstart einzulegen ist.

Der Autologon-Mechanismus ist abzustellen, da er die Sicherheitsmaßnahmen von SYSKEY aushöhlen würde.
Auch unter Windows erfolgt das Starten des Servers mit Administratorrechten. Die Anfragen werden durch einen zweiten Prozess bearbeitet, dessen Rechte wie unter Linux stark beschnitten sind. Dieser teilt die Anfragen einzelnen Threads zu, die die eigentliche Arbeit übernehmen [GSHB].
Ab Windows 2000 Server wird die Kerberos Authentifizierung genutzt, die gegenüber der früher angewendeten NTLM (NT-Lan-Manager) Authentifizierung mehr Sicherheit bietet. Kerberos benötigt keine umfangreiche Konfiguration und stellt nur wenige Konfigurationsparameter bereit. Diese können über die Gruppenrichtlinien angepasst werden. Der Kerberos KDC-Server (Key Distribution Center) prüft die Zugriffsberechtigung, die maximale Gültigkeitsdauer des Benutzertickets und des Diensttickets. Die hier eingestellte Zeitspanne ist die wesentliche Stellschraube für die Sicherheit der Anmeldung.
Da Diensten unter Windows standardmäßig die Berechtigung *Local System* zugeteilt wird, ist eine Einschränkung hier wichtig. Ein eigenes Dienstkonto bietet hier Abhilfe. Es muss aber getestet werden welche Rechte diesem Konto zugeteilt werden müssen, damit der Apache fehlerfrei betrieben werden kann.

Unter Sicherheitsaspekten spielt das Active Directory eine wichtige Rolle. Hierbei handelt es sich um eine verteilte Datenbank, die die Benutzer- und Konfigurationsdaten einer Domäne auf mehrere Domänen-Controller verteilt. Sie enthält viele sicherheitsrelevante Daten und die Basis für Zugriffsrechte und Privilegien in Windows 2000.

9. Sourcensicherheit

Der Quellcode des Apache Servers wird vorwiegend über das Internet verbreitet. Seltener findet eine Verbreitung über sichere Medien, wie CDs von kommerziellen Anbietern statt. Es wäre für Angreifer möglich eine Internetseite zu erzeugen, die die originale Quelltextseite von **`http://www.apache.org`** imitiert und hier Versionen anzubieten, die um Spionagemechanismen erweitert sind. Um diesem Risiko zu begegnen kann man die Originaldateien hashen und den Hashwert veröffentlichen. Nach dem Download kann man den Hashwert seines Downloads selbst berechen und mit dem Veröffentlichten vergleichen. Durch Veröffentlichung das Hashwertes des manipulierten Quellcodes kann dieser Sicherheitsmechanismus umgangen werden. Aus diesem Grund verwenden die Entwickler des Apache-Projekts digitale Signaturen die auf der Software PGP beruhen zur Absicherung des Codes. Diese arbeitet nach dem Public Key Verfahren und stellt die Integrität der Daten sicher.

10. Sicherheitslücken im Apache Webserver

Immer wieder werden Sicherheitslöcher im Apache gefunden, aber schon nach kurzer Zeit erkannt und ausgebessert.

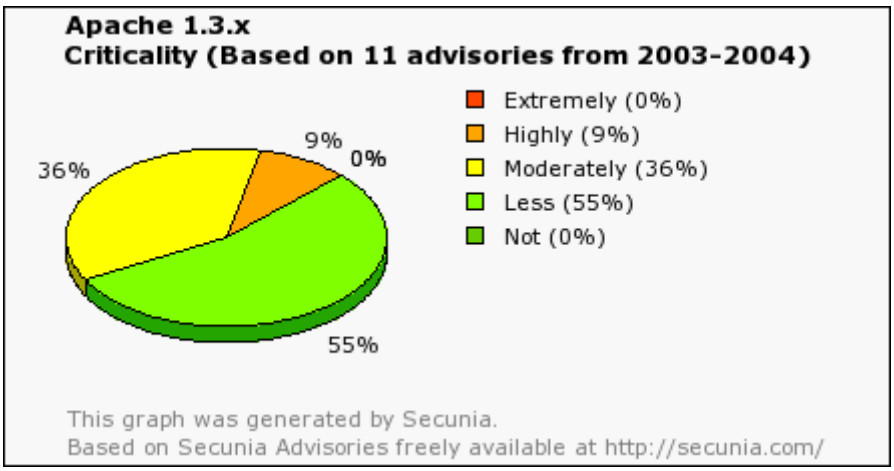

Abbildung 7: Secunia Bericht

Am 29.10.2004 wurde veröffentlicht, dass bei Apache v1.3.33 zwei Sicherheitslücken entdeckt wurden. Ein potentieller Buffer Overflow kann mit einem SSI Script ausgelöst werden wenn im *tag string* Buchstaben verloren gehen. Des Weiteren ist ein heap basierter Buffer Overflow in Modul proxy_util.c möglich. Durch einen negativen Wert im HTTP-Headerfeld *contentlength*, wird die Ausführung von Programmcode möglich. Eingestuft wird diese Sicherheitslücke als weniger gefährlich. Begründet wird es damit, dass der Prozess mit dem der Code ausgeführt wird im Regelfall kaum Privilegien besitz. Es wird vorgeschlagen, diese Sicherheitslücke durch einen Patch zu schließen.
Am 19.03.2004 wurde eine Sicherheitslücke in den Apacheversion 1.3.29 und früher entdeckt. Das Modul *mod-access* macht es Angreifern möglich, gesetzte Zugangsbeschränkungen zu umgehen. Allerdings besteht diese nur auf Plattformen die das Zahlenformat 64 bit big-endian verwenden.

Finden kann man diese Informationen über aktuell erkannte Sicherheitslücken im Internet unter den Adressen:
http://www.ciac.org/ciac/bulletins/o-221.shtml
http://www.cert.org/advisories
http://www.kb.cert.org/vuls
Um nicht immer selber aktiv werden zu müssen kann man sich in Maillisten eintragen, die einen über die neuerkannten Sicherheitslücken und geeignete Gegenmaßnahmen informieren.

11. Zusammenfassung

Die Sicherheit des Apache Webserver ist kein statischer Prozess. Bereits die Installation des Betriebssystems (Windows oder UNIX/Linux) bestimmt den späteren Sicherheitsstandard mit. Apache bietet mit seiner Standardauthentisierung eine bequeme

Möglichkeit Benutzerrechte zu vergeben. Die Unterstützung von SSL/TLS setzt neue Maßstäbe in der Sicherheit der Kommunikation. Es gibt aber keine Sicherheit aus der Dose: In weiten Teilen hat der Administrator für einen sicheren und geregelten Betrieb des Apache Webservers zu sorgen. Eine 100% sichere Konfiguration des Servers ist nicht möglich. Die Sicherheit wird weiterhin ein dynamischer Prozess sein, die bei unveränderter Konfiguration mit der Zeit abnimmt. Mailinglisten, die über erkannte Sicherheitslücken und Gegenmaßnahmen informieren, werden auch in Zukunft eine zentrale Rolle spielen. Notfallpläne sind eine wichtige Maßnahme zur Erhöhung der Serververfügbarkeit um, da sie helfen die Auswirkungen von unautorisierten Zugriffen zu minimieren.

12. Quellenverzeichnis

Eilebrecht Lars, Apache Web-Server, itp-Verlag, Bonn,1997

Laurie Ben, Laurie Peter, Apache Das umfassende Handbuch, 3. Auflage, O' Reilly, Köln, 2003

Siedler Kai, Vogelgesang Kay, Apache für Dummies, 1. korrigierter Nachdruck, itp-Verlag, Bonn, 2004

Siegler-Hornke Jens, Sollendick Wolfgang, Apache, Hanser, München Wien, 2001

http://wendtstud1.hpi.uni-potsdam.de/sysmod-seminar/elaborations/gruppe-4/mod_ssl.pdf
Sichere Kommunikation: Das Apachemodul „mod_ssl"

http://www.bsi-fuer-buerger.de/down/kurzvire.pdf
Computer-Viren: Definition und Wirkungsweise

http://de.wikipedia.org/wiki/MD5
MD 5 Algorithmus

http://httpd.apache.org
Homepage des Apache-Projektes

http://cert.uni-stuttgart.de/archive/win-sec-ssc/2001/08/msg00037.html
Sicherheitsschwachstelle in OpenSSL

http://www.microsoft.com/germany/windows/default.mspx
Informationen über Betriebssystem Windows

http://www.computer.de/Tips/Thema-frame.cfm?ID=7801
Funktionsweise SYSKEY

[CRL] http://www.apacheweek.com/features/crl *Apache Week: Using Certificate Revocation Lists*

[Auth] http://httpd.apache.org/docs/howto/auth.html *Apache Foundation: Authentication, Authorisation and Access Control*

[GSHB] IT-Grundschutzhandbuch, Bundeszentrale für Informationssicherheit (BSI), 2003

[mod_ssl] http://www.modssl.org/docs/2.8/ssl_overview.html mod_ssl: User Manual

13. Abbildungsverzeichnis