KAPITEL 11

Authentifizierung und Sicherheit

Bei vielen Webdatenbank-Anwendungen sind Einschränkungen erforderlich, um den Benutzerzugriff zu steuern. Einige Anwendungen arbeiten mit schützenswerten Informationnen wie Daten für Bankkonten. Andere bieten Informationen und Leistungen für zahlende Kunden. Bei derartigen Anwendungen ist es erforderlich, Benutzeranfragen zu authentifizieren und autorisieren. Das geschieht üblicherweise, indem ein Benutzername und ein Passwort eingelesen und anhand einer Liste gültiger Benutzer geprüft werden. Webanwendungen müssen nicht nur kontrolieren, wer Zugriff auf die Leistungen hat, sondern auch die Daten, die über das Internet übermittelt werden, vor denen schützen, die sie nicht sehen sollten.

In diesem Kapitel zeigen wir Ihnen die Techniken, die verwendet werden, um Webdatenbank-Anwendungen aufzubauen, die Benutzer zu authentifizieren und zu autorisieren und die Daten zu schützen, die über das Web übermittelt werden. Zu den Themen, die in diesem Kapitel behandelt werden, zählen:

- Die Funktionsweise der HTTP-Authentifizierung und ihre Verwendung mit Apache und PHP
- Das Schreiben von PHP-Skripten zur Authentifizierung und Autorisierung von Benutzern
- Die Autorisierung des Zugriffs über eine IP-Adresse oder einen Bereich von IP-Adressen
- Das Schreiben von PHP-Skripten zur Authentifizierung von Benutzern anhand einer Tabelle in einer Datenbank
- Praktische Aspekte des Aufbaus sessionbasierter Webdatenbank-Anwendungen zur Authentifizierung von Benutzern, einschließlich von Techniken, die keine HTTP-Authentifizierung einsetzen
- Eine Fallstudie, die ein Authentifizierungs-Framework entwickelt und viele der Techniken illustriert, die in diesem Kapitel vorgestellt wurden
- Die Features der Verschlüsselungsdienste, die das Secure Sockets Layer bietet

HTTP-Authentifizierung

Dieser Abschnitt setzt HTTP-Kenntnisse voraus. Wenn Sie damit nicht vertraut sind, finden Sie in Anhang D eine Einführung.

Der HTTP-Standard bietet Unterstützung für die Authentifizierung und die Autorisierung von Benutzerzugriffen. Wenn ein Browser einen HTTP-Request nach einer Ressource abschickt, für die eine Authentifizierung erforderlich ist, kann der Server auf die Anfrage mit einer HTTP-Response mit dem Statuscode `401 Unauthorized` antworten. Wenn der Browser eine Antwort erhält, die ihm sagt, dass die Anfrage nicht autorisiert war, zeigt er einen Dialog an, in dem ein Benutzername und ein Passwort eingegeben werden müssen. Wie dieser Dialog im Mozilla-Browser aussieht, zeigt Abbildung 11-1. Nachdem der Benutzername und das Passwort eingegeben worden sind, schickt der Browser die ursprüngliche Anfrage erneut ab und schließt darin ein zusätzliches Header-Feld ein, das die Benutzerberechtigungen kodiert.

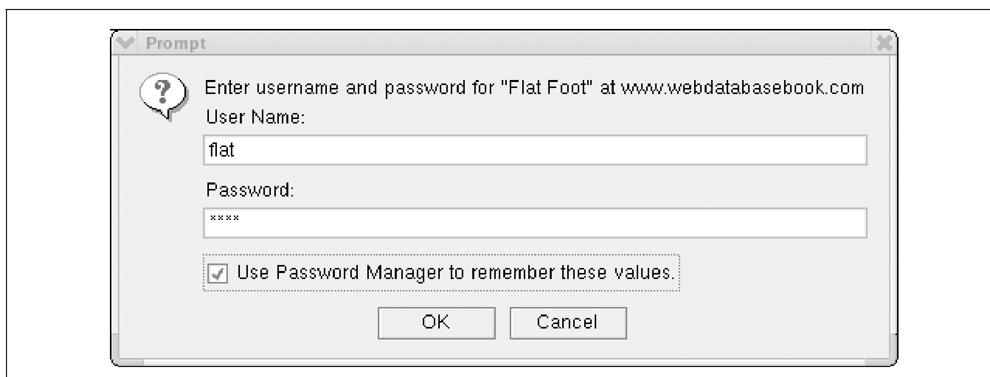

Abbildung 11-1: Mozilla verlangt nach einem Benutzernamen und einem Passwort.

Der HTTP-Header sammelt bloß den Namen und das Passwort. Er sorgt weder für die Authentifizierung des Benutzers noch bietet er die Autorisierung für den Zugriff auf eine Ressource oder einen Dienst. Der Server muss die kodierten Benutzername- und Passwort-Informationen verwenden, um zu entscheiden, ob der Benutzer für den Zugriff auf die angeforderte Ressource autorisiert ist. Beispielsweise könnten Sie Ihren Apache-Webserver so konfigurieren, dass er eine Authentifizierung verlangt, die anhand einer Datei mit einer Liste mit Benutzernamen und verschlüsselten Passwörtern erfolgt. In einer anderen Anwendung könnten Sie eine Tabelle mit Benutzernamen und Passwörtern einsetzen, die in einer Datenbank gespeichert wird, und dann PHP-Code für den Authentifizierungsprozess entwickeln.

Wie eine HTTP-Authentifizierung funktioniert

Abbildung 11-2 zeigt die Interaktion zwischen einem Webbrowser und einem Webserver, wenn ein HTTP-Request eine Authentifizierungsaufforderung auslöst. Der Benutzer

verlangt nach einer auf dem Server gespeicherten Ressource, für die eine Authentifizierung erforderlich ist. Der Server schickt eine HTTP-Response mit dem Statuscode `401 Unauthorized`, auf die die Anfrage mit einer Authentifizierungsaufforderung antwortet. In diese Antwort ist das Header-Feld `WWW-Authenticate` eingeschlossen, das die Parameter enthält, die dem Browser mitteilen, wie er auf die Authentifizierungsaufforderung reagieren muss. Es kann sein, dass der Browser zur Eingabe eines Benutzernamens und eines Passworts auffordern muss, um der Authentifizierungsaufforderung Genüge zu tun. Dann schickt der Browser die Anfrage erneut ab und schließt dabei das Header-Feld `Authorization` ein, das die Berechtigungen enthält, die der Server benötigt.

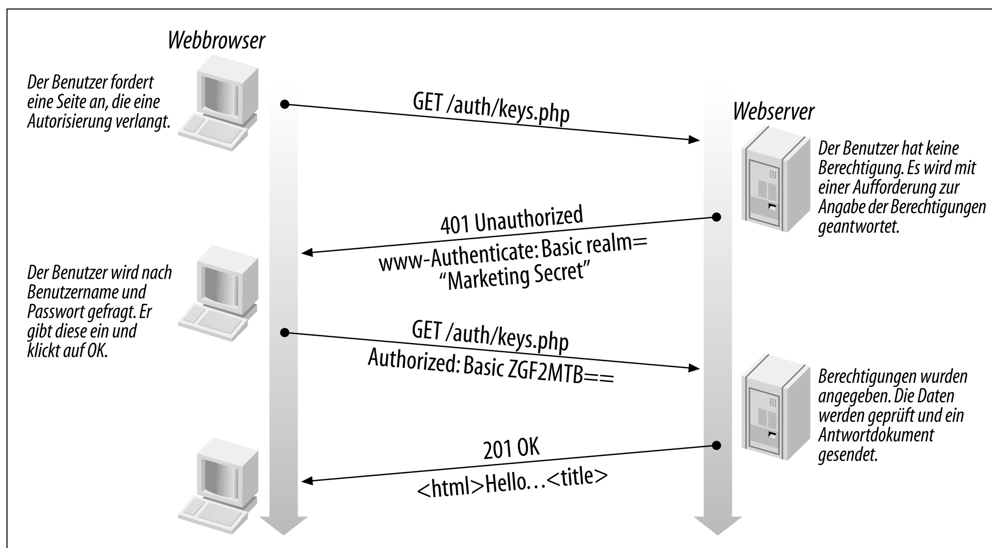

Abbildung 11-2: Die Abfolge der HTTP-Requests und -Responses bei Anfrage einer Seite, für die eine Authentifizierung erforderlich ist

Hier sehen Sie ein Beispiel für eine HTTP-Response, die ein Apache-Server zurückschickt, wenn ein HTTP-Request nach einer Resource erfolgt, für die eine Authentifizierung erforderlich ist:

```
HTTP/1.1 401 Authorization Required
Date: Thu, 2 Dec 2004 23:40:54 GMT
Server: Apache/2.0.48 (Unix) PHP/5.0.0
WWW-Authenticate: Basic realm="Marketing Secret"
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html401/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>401 Authorization Required</title>
</head>
```

```
<body>
<h1>Authorization Required</h1>
This server could not verify that you
are authorized to access the document
requested. Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.
<p><hr>
</body>
</html>
```

Das Header-Feld `WWW-Authenticate` enthält die *Authentifizierungsmethode*, die dem Browser mitteilt, wie er die Benutzerberechtigungen sammeln und kodieren soll. In diesem Beispiel ist die Methode `Basic`. Der Header enthält außerdem den Namen des *Bereichs* (Realm), auf den die Authentifizierung Auswirkungen hat. Hier ist das *Marketing Secret*. Der Bereich wird vom Browser als Schlüssel für das Benutzername/Passwort-Paar verwendet und außerdem angezeigt, wenn die Berechtigungen aufgenommen werden.

Abbildung 11-1 zeigt den Dialog, der für den Bereich *Marketing Secret* angezeigt wird. Nachdem der Browser die Berechtigung vom Benutzer entgegengenommen hat, sendet er die ursprüngliche Anfrage mit dem zusätzlichen Header-Feld `Authorization` ab, das die Berechtigungen enthält. Hier sehen Sie ein Beispiel für einen HTTP-Request, der im Header-Feld `Authorization` die Berechtigungen enthält:

```
GET /auth/keys.php HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4.51 [en] (WinNT; I)
Host: localhost
Accept: image/gif, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Authorization: Basic ZGF2ZTpwbGF0eXB1cw==
```

Ein Browser kann automatisch auf die Authentifizierungaufforderung antworten, wenn die Berechtigungen für diesen Bereich bereits gesammelt worden sind. Er fährt fort, die Authentifizierungsberechtigungen in Anfragen einzuschließen, bis das Browser-Programm beendet oder ein anderer Bereich betreten wird.

Bei der Codierungsmethode `Basic` werden der Benutzername und das Passwort im Header-Feld `Authorization` versandt, nachdem sie mit Base-64 codiert wurden. Die Base-64-Codierung soll die Daten nicht schützen und stellt keine Form der Verschlüsselung dar. Sie sorgt nur dafür, dass binäre Daten über ein Netzwerk verschickt werden. Im günstigsten Fall schützt sie die Daten nur davor, einfach so eingesehen zu werden.

Einige Webserver, einschließlich Apache, unterstützen die Codierungsmethode `Digest`. Die Methode `Digest` ist sicherer als `Basic`, weil das Passwort des Benutzers nicht über das Netzwerk verschickt wird. Damit sie verwendet werden kann, muss der Browser sie aber ebenfalls unterstützen. Die wichtigeren Browser, die die `DIGEST`-Methode unterstützen,

sind Opera, Microsoft Internet Explorer, Amaya, Mozilla und Netscape. Weil die DIGEST-Methode nicht so weit verbreitet implementiert ist, sollten Sie sie deswegen nur verwenden, wenn Sie Kontrolle über die Browser-Wahl Ihrer Benutzer haben.

Während die Codierungsmethode Basic keine wirkliche Sicherheit bietet, kann das SSL-Protokoll (Secure Sockets Layer-Protokoll) die HTTP-Requests und -Responses schützen, die zwischen dem Browser und dem Server verschickt werden. Das bedeutet, dass SSL auch Schutz für die Benutzernamen und Passwörter bietet, die mit der Basic-Methode übermittelt werden. Deswegen empfehlen wir die Verwendung von SSL bei Webdatenbank-Anwendungen, die schützenswerte Informationen übermitteln. SSL werden wir weiter unten in diesem Kapitel behandeln.

Apache zur Authentifizierung verwenden

Die einfachste Methode, den Zugriff auf eine Anwendung einzuschränken, ist die Verwendung der eingebauten Authentifizierungsunterstützung Ihres Webservers. Der Apache-Webserver kann problemlos so konfiguriert werden, dass zum Schutz der Ressourcen des Servers eine HTTP-Authentifizierung verwendet wird. Beispielsweise ermöglicht Apache es, eine Authentifizierung auf Verzeichnisebene einzurichten, indem der Directory-Einstellung in der Konfigurationsdatei *httpd.conf* Parameter hinzugefügt werden.

Das folgende Beispiel zeigt einen Ausschnitt aus einer *httpd.conf*-Datei, die Ressourcen (wie HTML-Dateien, PHP-Skripten, Bilder usw.) schützt und im Verzeichnis */usr/local/apache/htdocs/auth* gespeichert wird:

```
# Set up an authenticated directory
<Directory "/usr/local/apache/htdocs/auth">
  AuthType Basic
  AuthName "Secret Mens Business"
  AuthUserFile /usr/local/apache/allow.users
  require hugh, dave, jim
</Directory>
```

Wenn Sie Microsoft Windows verwenden, können Sie */usr/local/apache/htdocs/auth* durch ein Verzeichnis wie *C:\Programme\EasyPHP1-7\www\auth* ersetzen. Auf einer Mac OS X-Plattform verwenden Sie ein Verzeichnis wie */Library/WebServer/Documents/auth*. Auf alle Fälle muss das *auth*-Verzeichnis vorhanden sein.

Ein Benutzer muss an der Apache-Authentifizierung vorbei, bevor ihm Zugriff auf Ressourcen – einschließlich der PHP-Skripten – gestattet wird, die in einem Verzeichnis gespeichert sind, für das eine Authentifizierung erforderlich ist. Auf nicht-authentifizierte Anfragen nach Ressourcen im geschützten Verzeichnis antwortet der Apache-Server mit einer Authentifizierungsaufforderung. Der Authentifizierungstyp wird über den Parameter AuthType auf Basic gesetzt, um die Methode anzugeben, wie der Benutzername und das Passwort vom Benutzer entgegengenommen werden. Der Parameter AuthName wird auf den Namen des Bereichs gesetzt. Apache autorisiert die Benutzer, die in der require-Einstellung aufgeführt werden, indem der Benutzername und das Passwort mit denen

verglichen werden, die in der Datei aufgeführt sind, die nach der Direktive `AuthUserFile` angegeben wird. Die anderen Parameter werden hier nicht besprochen. Sie sollten sich die Apache-Referenzen ansehen, die in Anhang G aufgeführt werden, wenn Sie vollständige Informationen zur Konfiguration suchen.

Wenn Sie keinen Administrator- oder Root-Zugriff auf Ihren Webserver-Rechner haben, können Sie trotzdem ein Verzeichnis (oder ausgewählte Ressourcen in einem Verzeichnis) schützen. Das erreichen Sie, indem Sie in dem Verzeichnis, das Sie schützen wollen, eine *.htaccess*-Datei anlegen und darin angeben, welche Ressourcen geschützt sind, wer auf sie zugreifen darf und wo die Passwörter zu finden sind. Es ist einfach, mit PHP Ressourcen zu schützen – wie wir im nächsten Abschnitt besprechen werden –, aber diese Vorgehensweise werden wir hier nicht detailliert behandeln. Mehr Informationen finden Sie unter *http://httpd.apache.org/docs-2.0/howto/htaccess.html*.

Bei vielen Webdatenbank-Anwendungen bietet die Apache-Authentifizierung eine einfache Lösung. Wenn Benutzernamen und Passwörter allerdings anhand einer Datenbank geprüft werden müssen oder die HTTP-Authentifizierung den Anforderungen einer Anwendung nicht genügt, kann die Authentifizierung stattdessen mit PHP gesteuert werden. Der nächste Abschnitt beschreibt, wie PHP die HTTP-Authentifizierung direkt und ohne Konfiguration von Apache steuern kann. Später beschreiben wir auch, wie man eine Authentifizierung bieten kann, ohne HTTP zu verwenden.

HTTP-Authentifizierung mit PHP

Mit selbst geschriebenen PHP-Skripten zur Steuerung des Authentifizierungsvorgangs lässt sich eine flexible Autorisierungslogik gestalten. Beispielsweise könnte eine Anwendung auf Basis von Gruppenmitgliedschaften Einschränkungen anwenden: Ein Benutzer aus der Finanzabteilung kann Berichte aus der Etat-Datenbank abrufen, andere hingegen können das nicht. In einer anderen Anwendung kann einem Benutzer eines subskriptionsbasierten Dienstes, der eine richtige Benutzername/Passwort-Kombination angibt, der Zugriff verweigert werden, wenn die Zahlung seit vierzehn Tagen überfällig ist. Oder der Zugriff könnte jeden Donnerstagabend der Australian Eastern Standard Time verweigert werden, weil dann das System gewartet wird.

PHP-Skripten verleihen Ihnen mehr Kontrolle über den Authentifizierungvorgang als Apache-Dateien oder die Apache-Konfiguration. In diesem Abschnitt zeigen wir Ihnen, wie PHP-Skripten Authentifizierungsberechtigungen verwenden können und wie Sie einfache, flexible Authentifizierungsskripten entwickeln, die HTTP verwenden.

Auf Benutzerberechtigungen zugreifen

Wenn PHP eine Anfrage verarbeitet, die im Header-Feld `Authorized` codierte Berechtigungen enthält, ist über die superglobale Variable `$_SERVER` der Zugriff auf diese Berechtigungen möglich. Das Element `$_SERVER["PHP_AUTH_USER"]` enthält den Benutzernamen, den der Benutzer angegeben hat, und `$_SERVER["PHP_AUTH_PW"]` enthält das Passwort.

Das in Beispiel 11-1 gezeigte Skript liest die superglobalen Variablen für die Authentifizierung und zeigt sie im Body der Antwort an. In der Praxis würden Sie sie dem Benutzer natürlich nicht wieder anzeigen, weil das unsicher ist – hier tun wir das nur, um zu demonstrieren, wie auf sie zugegriffen werden kann. Stattdessen würden Sie die Berechtigungen verwenden, um den Benutzer zu authentifizieren und ihm den Zugriff auf die Anwendung zu erlauben oder zu verweigern. Wie Sie das tun, erläutern wir im nächsten Abschnitt.

Damit mit dem PHP-Code in Beispiel 11-1 die Authentifizierungsberechtigungen angezeigt werden können, muss das Skript erneut angefragt werden, nachdem der Benutzer zur Eingabe seines Benutzernamens und Passworts aufgefordert wurde. Beispielsweise kann die Authentifizierungsaufforderung ausgelöst werden, indem die Skriptdatei in ein Verzeichnis gesteckt wird, das mit Apache so konfiguriert wurde, dass für es eine Authentifizierung erforderlich ist, wie wir es im vorangegangenen Abschnitt besprochen haben. Die Verwendung der superglobalen Variablen löst die Authentifizierung nicht aus, sie bietet einfach nur Zugriff auf die Werte, die der Benutzer angegeben hat.

Beispiel 11-1: PHP-Zugriff auf die Authentifizierung

```
<!DOCTYPE HTML PUBLIC
                "-//W3C//DTD HTML 4.01 Transitional//EN"
                "http://www.w3.org/TR/html401/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Authentication</title>
</head>
<body>
<?php
  if (isset($_SERVER["PHP_AUTH_USER"]))
    print "<h2>Hi there {$_SERVER["PHP_AUTH_USER"]}</h2>";
  else
    print "You need to be authenticated for this to work!";

  if (isset($_SERVER["PHP_AUTH_PW"]))
    print "<p>Thank you for your password {$_SERVER["PHP_AUTH_PW"]}!";
?>
</body>
</html>
```

Über den Zugriff auf Informationen im Header-Feld für die Authentifizierung können einfache Anwendungen entwickelt werden, bei denen eine Identifikation des Benutzers erforderlich ist. Beispielsweise könnte eine Anwendung, die Zahlungen auf Seitenbasis fordert, die Variable `$_SERVER["PHP_AUTH_USER"]` einsetzen, um den Zugriff auf eine bestimmte Seite aufzuzeichnen. Auf diese Weise kann Apache die Authentifizierung bieten und die Anwendung das Verhalten des Benutzers aufzeichnen.

Auch wenn es bei diesem einfachen Verfahren zur Entwicklung von Anwendungen nicht notwendig ist, irgendwelchen PHP-Code zur Implementierung der Authentifizierung zu

schreiben, müssen die Benutzer und die Passwörter in einer Apache-Passwort-Datei gewartet werden. Im nächsten Abschnitt beschreiben wir, wie man die HTTP-Authentifizierung aus einem PHP-Skript heraus steuert und so Apache von der Authentifizierungsverantwortung befreit und es ermöglicht, auf die Anfrage-Autorisierung eine komplexere Logik anzuwenden.

HTTP-Authentifizierung mit PHP verwalten

PHP-Skripten können HTTP-Authentifizierungsaufforderungen verwalten. Dazu prüfen Sie, ob die Variablen `$_SERVER["PHP_AUTH_USER"]` und `$_SERVER["PHP_AUTH_PW"]` gesetzt sind. Sind sie nicht gesetzt, wurde der Benutzer nicht authentifiziert. Dann senden Sie eine Antwort an den Browser zurück, die den `WWW-Authenticate`-Header enthält. Wenn die Variablen gesetzt sind, ist der Benutzer der Authentifizierungsaufforderung nachgekommen. Dann prüfen Sie sie im Skript mit der erforderlichen Logik anhand der gespeicherten Berechtigungen. Wenn die Berechtigungen denen entsprechen, die im Skript gespeichert sind, wird dem Benutzer der Zugriff auf das Skript gestattet. Wenn nicht, wird erneut eine Authentifizierungsaufforderung an den Browser gesandt.

In Beispiel 11-2 werden die Benutzerberechtigungen an die Funktion *authenticated()* übergeben. Diese Funktion ist ein einfaches Authentifizierungsschema, um zu prüfen, dass das Passwort dem entspricht, das in das Skript hartcodiert wurde. Stimmen beide überein, wird dem Benutzer Zugriff auf die Anwendung gestattet. Um das Skript zu testen, wählen Sie einen beliebigen Benutzernamen und das Passwort `kwAlIphIdE` (Groß-/Kleinschreibung ist wichtig). Das Template, das mit dem Beispiel verwendet wird, wird in Beispiel 11-3 gezeigt.

Beispiel 11-2: Ein Skript zur Erzeugung einer »Nicht-Autorisiert«-Antwort

```
<?php
require_once "HTML/Template/ITX.php";
require "db.inc";

function authenticated($username, $password)
{
  // Wenn Benutzername oder Passwort nicht gesetzt sind,
  // wurde der Benutzer nicht authentifiziert
  if (!isset($username) || !isset($password))
    return false;

  // Ist das Passwort richtig?
  // Wenn ja, ist der Benutzer authentifiziert
  if ($password == "kwAlIphIdE")
    return true;
  else
    return false;
}

$template = new HTML_Template_ITX("./templates");
$template->loadTemplatefile("example.11-3.tpl", true, true);
```

Beispiel 11-2: Ein Skript zur Erzeugung einer »Nicht-Autorisiert«-Antwort (Fortsetzung)

```
$username = shellclean($_SERVER, "PHP_AUTH_USER", 20);
$password = shellclean($_SERVER, "PHP_AUTH_PW", 20);

if(!authenticated($username, $password))
{
  // Keine Berechtigungen gefunden.
  // Antwort mit Authentifizierungsaufforderung versenden
  header("WWW-Authenticate: Basic realm=\"Flat Foot\"");
  header("HTTP/1.1 401 Unauthorized");

  // Den Body der Anwort einrichten, der angezeigt wird,
  // wenn der Benutzer die Authentifizierungsaufforderung abbricht
  $template->touchBlock("challenge");
  $template->show( );
  exit;
}
else
{
  // Den authentifizierten Benutzer begrüßen
  $template->touchBlock("authenticated");
  $template->show( );
}
?>
```

Beispiel 11-3: Das mit Beispiel 11-2 verwendete Template

```
<!DOCTYPE HTML PUBLIC
                "-//W3C//DTD HTML 4.01 Transitional//EN"
                "http://www.w3.org/TR/html401/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Web Database Applications</title>
</head>
<body>
<!-- BEGIN challenge -->
  <h2>You need a username and password to access this service</h2>
  <p>If you have lost or forgotten your password, tough!
<!-- END challenge -->
<!-- BEGIN authenticated -->
  <h2>Welcome!</h2>
<!-- END authenticated -->
</body>
</html>
```

Die Funktion *authenticated()* liefert `false` zurück, wenn `$username` oder `$password` nicht gesetzt sind oder das Passwort nicht der Zeichenfolge `kwAlIphIdE` entspricht. Wenn die Benutzerberechtigungen den Test nicht bestehen, antwortet das Skript mit dem Header-Feld `WWW-Authenticate` und setzt dabei das Codierungsschema auf `Basic` und den Bereich auf `Flat Foot`. Es schließt auch den Statuscode `401 Unauthorized` ein. Das PHP-Manual

schlägt vor, die Antwortzeile WWW-Authenticate vor der Antwortzeile HTTP/1.1 401 Unauthorized zu senden, um Probleme mit einigen älteren Versionen des Internet Explorers zu vermeiden.

Wenn ein Browser zum ersten Mal diese Seite verlangt, sendet das Skript als Antwort eine Autorisierungsaufforderung, die das Header-Feld 401 Unauthorized enthält. Wenn der Benutzer die Authentifizierungsaufforderung abbricht, indem er z.B. auf den ABBRECHEN-Button des Dialogfensters klickt, in dem die Berechtigungen eingegeben werden sollen, wird das HTML angezeigt, das in der HTTP-Response mit der Authentifizierungsaufforderung kodiert ist. Gibt der Benutzer die richtigen Berechtigungen an (einen Benutzernamen und das Passwort kwAlIphIdE), wird eine Willkommensnachricht angezeigt. Gibt der Benutzer falsche Berechtigungen an, ohne auf ABBRECHEN zu klicken, wird der Authentifizierungsdialog so lange erneut angezeigt, bis der Benutzer die richtigen Berechtigungen eingibt oder auf ABBRECHEN klickt.

Zugriff auf bestimmte IP-Adressen beschränken

Manchmal ist es nützlich, den Zugriff auf eine Anwendung oder einen Teil einer Anwendung auf Benutzer zu beschränken, die Teil eines bestimmten Netzwerks sind oder einen bestimmten Rechner verwenden. Beispielsweise könnte der Zugriff auf administrative Funktionen auf einen einzelnen Rechner beschränkt oder der Zugriff auf die neueste Version Ihrer Anwendung nur Mitgliedern des Test-Teams gestattet werden. Diese Typen von Einschränkungen lassen sich in PHP recht einfach implementieren. Die IP-Adresse des Rechners, von dem ein HTTP-Request ausging, können Sie über die Variable $_SERVER["REMOTE_ADDR"] ermitteln. Dasselbe können Sie auch mit Apache tun. Aber das werden wir hier nicht besprechen. (IP-Adressen können auch eingesetzt werden, um dem Kapern von Sessions vorzubeugen. Das ist ein Problem, das wir später in diesem Kapitel besprechen werden.)

Das in Beispiel 11-4 gezeigte Skript gestattet Benutzern den Zugriff, deren Rechner sich in einem bestimmten Subnetz eines Netzwerks befinden. Das Skript schränkt den Zugriff auf den Hauptinhalt des Skripts auf Anfragen ein, die von Clients ausgeführt werden, die sich in einem Bereich von IP-Adressen befinden, der mit 141.190.17 beginnt. Weil das bloß der Anfang einer Adresse ist, prüfen wir nur die ersten 10 Zeichen. Das Template, das mit dem Beispiel verwendet wird, wird in Beispiel 11-5 gezeigt.

Beispiel 11-4: Ein PHP-Skript, das einen Zugriff von Browsern außerhalb eines IP-Subnetzes unterbindet

```
<?php
require_once "HTML/Template/ITX.php";

$template = new HTML_Template_ITX("./templates");
$template->loadTemplatefile("example.11-5.tpl", true, true);
```

Beispiel 11-4: Ein PHP-Skript, das einen Zugriff von Browsern außerhalb eines IP-Subnetzes unterbindet (Fortsetzung)

```
if(strncmp("141.190.17", $_SERVER["REMOTE_ADDR"], 10) != 0)
{
  // Nicht zulässig
  header("HTTP/1.1 403 Forbidden");
  $template->touchBlock("noaccess");
  $template->show( );
  exit;
}
else
{
  // Zulässig
  $template->touchBlock("authenticated");
  $template->show( );
}
?>
```

Beispiel 11-5: Das mit Beispiel 11-4 verwendete Template

```
<!DOCTYPE HTML PUBLIC
                 "-//W3C//DTD HTML 4.01 Transitional//EN"
                 "http://www.w3.org/TR/html401/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Web Database Applications</title>
</head>
<body>
<!-- BEGIN noaccess -->
<h2>403 Forbidden</h2>
<p>You cannot access this page from outside the Marketing Department.
<!-- END noaccess -->
<!-- BEGIN authenticated -->
<h2>Marketing secrets!</h2>
<p>Need new development team - the old one says <i>No</i> far too often.
<!-- END authenticated -->
</body>
</html>
```

Es gibt verschiedene HTTP-Statuscodes, die geeignet sind, einem Benutzer den Zugriff zu verweigern. Im vorangegangenen Abschnitt haben wir den Antwortcode `401 Unauthorized` verwendet, um die HTTP-Authentifizierung zu steuern. Der Statuscode `403 Forbidden` ist in einer Antwort aber geeigneter, um zu erklären, warum der Zugriff verweigert wurde. Deswegen wird er in Beispiel 11-4 verwendet. Der Standard HTTP/1.1 beschreibt 17 `4xx` Statuscodes, die unterschiedliche Bedeutungen haben. Der berüchtigte Status `404 Not Found` wird von Apache zurückgeliefert, wenn eine angefragte Ressource nicht existiert. Ein PHP-Skript kann diesen Code zurückliefern, wenn der genaue Grund für die Ablehnung der Anfrage verborgen bleiben soll.

Authentifizierung mit Hilfe einer Datenbank

In diesem Abschnitt zeigen wir Ihnen, wie Skripten eine Authentifizierung durchführen können, indem Sie eine Datenbank-Tabelle abfragen, die Benutzernamen und Passwörter enthält. Weil die Berechtigungen der Benutzer schützenswerte Daten sind, zeigen wir Ihnen, wie Sie Passwörter mit einer Verschlüsselung schützen und wie das verschlüsselte Passwort im Authentifizierungsvorgang verwendet wird.

Eine Datenbank und eine Tabelle anlegen

Um die Prinzipien der Verwendung einer Datenbank zur Verwaltung der Authentifizierung zu demonstrieren, benötigen wir eine Tabelle, in der Benutzernamen und Passwörter gespeichert werden können, und einen Benutzer, der auf die Datenbank und die Tabelle zugreifen kann. Hierbei müssen Sie unbedingt beachten, dass das zwei verschiedene Dinge sind. Die Datenbank-Tabelle wird verwendet, um die Benutzernamen und Passwörter für die Benutzer unserer Anwendung zu speichern. Der MySQL-Benutzer hingegen wird in unseren PHP-Skripten nur verwendet, um Daten aus der Datenbank zu lesen oder in sie zu schreiben. In diesem Abschnitt richten wir die Datenbank, die Tabelle und einen MySQL-Account ein.

In den restlichen Beispielen in diesem Kapitel verwenden wir die Datenbank *authentication*, die eine Tabelle *users* enthält. Sie müssen sich als MySQL-Root-User einloggen, um beide anzulegen, indem Sie Folgendes im MySQL-Befehlsinterpreter eingeben:

```
mysql> create database authentication;
Query OK, 1 row affected (0.05 sec)

mysql> use authentication;
Database changed
mysql> CREATE TABLE users (
    ->   user_name char(50) NOT NULL,
    ->   password char(32) NOT NULL,
    ->   PRIMARY KEY (user_name)
    -> ) type=MyISAM;
Query OK, 0 rows affected (0.02 sec)
```

Die *users*-Tabelle definiert zwei Attribute: `user_name` und `password`. Der `user_name` muss eindeutig sein und ist als der Primärschlüssel definiert.

Es muss auch einen MySQL-Benutzer geben, der Zugriff auf diese Datenbank hat. Mit folgender Anweisung, die Sie wieder im MySQL-Befehlsinterpreter eingeben, können Sie eine Benutzerin `lucy` mit dem Passwort `secret` anlegen:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON authentication.users TO
    -> lucy@127.0.0.1 IDENTIFIED BY 'secret';
Query OK, 0 rows affected (0.00 sec)
```

Die Syntax dieser Anweisung wird in Kapitel 15 besprochen. Die Benutzerin `lucy` werden wir in den Skripten im restlichen Kapitel verwenden.

Passwörter schützen

Es stellt ein Sicherheitsrisiko dar, wenn die Passwörter von Benutzern im Klartext gespeichert werden. Insider, externe Hacker und andere können sich über sie Zugriff auf eine Datenbank verschaffen. Deswegen ist es üblich, Passwörter mit einem nicht-umkehrbaren Einweg-Verschlüsselungsalgorithmus zu verschlüsseln und die verschlüsselte Version in der Datenbank zu speichern. Diese verschlüsselte Version wird dann im Authentifizierungsprozess verwendet. (Die Einweg-Verschlüsselung oder asymmetrische Verschlüsselung wird später in diesem Kapitel behandelt.)

Der Prozess des Schützens von Passwörtern läuft folgendermaßen ab: Zunächst werden vom Benutzer ein neuer Benutzername und ein neues Passwort entgegengenommen. Dann wird das Passwort verschlüsselt und eine neue Zeile in die Tabelle *users* eingefügt, die den Klartext-Benutzernamen und das verschlüsselte Passwort enthält. Wenn der Benutzer später zurückkehrt und sich in die Anwendung einloggen will, gibt er seinen Benutzernamen und sein Passwort ein. Das Passwort, das der Benutzer eingegeben hat, wird dann verschlüsselt, und aus der Tabelle *users* wird die Zeile abgerufen, die dem angegebenen Benutzernamen entspricht. Dann wird die verschlüsselte Version des Passworts, das der Benutzer angegeben hat, mit der verschlüsselten Version verglichen, die in der Tabelle gespeichert ist. Wenn der Benutzername und das verschlüsselte Passwort zusammenpassen, sind die Berechtigungen korrekt und der Benutzer wurde erfolgreich authentifiziert.

PHP bietet zwei Funktionen, die zur Einweg-Verschlüsselung von Passwörtern verwendet werden können. Wir werden erst die Funktionen beschreiben und Ihnen dann Beispiele vorstellen, die ihr Verhalten ausführlicher erläutern.

string crypt(string `message [, string salt]`)

Auf den meisten Plattformen liefert diese Funktion einen verschlüsselten String zurück, der mit einem beliebten (wenn auch etwas älteren) Verschlüsselungsalgorithmus berechnet wurde, der *DES* heißt. Der zu verschlüsselnde Klartext *`message`* wird als erstes Argument angegeben. Das optionale zweite Argument, das Salt, wird verwendet, um den DES-Verschlüsselungsalgorithmus zu *salzen*. Standardmäßig werden die ersten acht Zeichen von *`message`* verschlüsselt. Der *`Salt`* besteht aus zwei Zeichen, die von DES verwendet werden, um das Knacken des verschlüsselten Strings zu erschweren. Wenn kein Salt angegeben wurde, erzeugt PHP einen zufälligen Wert. Die ersten beiden Zeichen des zurückgelieferten Werts entsprechen dem Salt, das im Verschlüsselungsprozess verwendet wurde.

Wie wir später zeigen werden, wird ein Salt verwendet, um zu verhindern, dass zwei Passwörter, die identisch sind, zu ein und demselben String verschlüsselt werden. Beide, Salt und Passwort, gehen in die Verschlüsselungsfunktion ein. Wenn zwei Passwörter identisch sind, das Salt aber verschieden ist, dann ist deswegen auch die Ausgabe anders. Soll für einen Vergleich mit einem verschlüsselten String ein anderer String verschlüsselt werden, müssen Sie wissen, welches Salt verwendet wurde,

damit Sie es erneut verwenden können. Aus diesem Grund wird das Salt als die ersten beiden Zeichen des verschlüsselten Strings zurückgeliefert.

Das ist eine Einweg-Funktion: Der zurückgelieferte Werte kann nicht wieder in den Ausgangsstring entschlüsselt werden.

Verschiedene PHP-Konstanten steuern den Verschlüsselungsvorgang. Bei der Beschreibung, die wir gerade geliefert haben, wird das Standardverhalten vorausgesetzt. Auf einigen Plattformen wird von den Interna der Funktion eigentlich aber der MD5-Ansatz verwendet, der als Nächstes behandelt wird, oder das Salt kann länger sein. Mehr Informationen finden Sie im PHP-Manual.

string md5(string message)

Liefert eine 32 Zeichen lange *Nachrichtenprüfsumme*, der mit dem MD5 Message Digest Algorithm der Firma RSA Data Security, Inc. (*http://www.faqs.org/rfcs/rfc1321.html*.) aus der Quelle *message* berechnet wird. Eine Prüfsumme ist ein 32 Zeichen langer Fingerabdruck bzw. eine Signatur und ist keine verschlüsselte Repräsentation der Nachricht selbst. Die MD5-Nachrichtenprüfsumme wird berechnet, indem die ganze Nachricht untersucht wird. Nachrichten, die sich bloß durch ein einzelnes Zeichen unterscheiden, bringen sehr verschiedene Prüfsummenergebnisse hervor. Wie die *crypt()*-Funktion ist auch *md5()* eine Einweg-Funktion.

Es ist unmöglich, aus einer Prüfsumme die ursprüngliche Nachricht zu rekonstruieren. Die Prüfsumme einer Nachricht ist immer 32 Zeichen lang und keine verschlüsselte Repräsentation der Nachricht. Stattdessen ist es ein String, der aus der Nachricht berechnet wird und fast sicher für diese Nachricht eindeutig ist.

Diese Funktion wird auf den meisten Plattformen verbreitet unterstützt und sollte bei Code, der portabel sein soll, anstelle von *crypt()* verwendet werden. Beachten Sie, dass MD5-Prüfsummen und die `Digest`-Authentifizierung von Apache nichts miteinander zu tun haben.

Beispiel 11-6 zeigt, wie *crypt()* und *md5()* eingesetzt werden. Die Skripten erzeugen die folgende Ausgabe:

```
md5(aardvark7) = 94198c7f71931fdeb0a7f4b75a603586
crypt(aardvark7, 'aa') = aaE/1j3.OKy/Y
crypt(aardvark7, 'bb') = bbptug8K4z6vA

md5(aardvark8) = 4a68f92613baa5202d523134e768db13
crypt(aardvark8, 'aa') = aaE/1j3.OKy/Y
crypt(aardvark8, 'bb') = bbptug8K4z6vA
```

Beispiel 11-6: crypt() und md5() verwenden

```
<!DOCTYPE HTML PUBLIC
               "-//W3C//DTD HTML 4.01 Transitional//EN"
               "http://www.w3.org/TR/html401/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Passwords</title>
```

Beispiel 11-6: crypt() und md5() verwenden (Fortsetzung)

```
</head>
<body>
<?php
$passwords = array();
$passwords[] = "aardvark7";
$passwords[] = "aardvark8";

foreach($passwords as $password)
{
  print "\n<p> md5({$password}) = " . md5($password);
  print "\n<br> crypt({$password}, 'aa') = " . crypt($password, "aa");
  print "\n<br> crypt({$password}, 'bb') = " . crypt($password, "bb");
}
?>
</body>
</html>
```

Beide Funktionen haben Vorteile und Nachteile:

- *md5()* funktioniert mit Strings beliebiger Länge. Es liefert einen String mit einer festgelegten Länge von 32 Zeichen zurück, der bei unterschiedlichen Eingabe-Strings jeweils unterschiedlich ist. Wie man erwarten würde, sind die Ergebnisse für `aardvark7` und `aardvark8` in Beispiel 11-6 unterschiedlich.
- *crypt()* verwendet nur die ersten acht Zeichen eines Passworts und ein Salt, um den verschlüsselten String zu berechnen. Wenn die ersten acht Zeichen und das Salt gleich sind, sind deswegen auch die beiden verschlüsselten Strings gleich. In Beispiel 11-6 gibt es keinen Unterschied zwischen den Ergebnissen für `aardvark7` und `aardvark8`, wenn das Salt gleich ist.
- Bei *crypt()* bringt das Salt ein praktisches Zusatz-Feature ein, das von *md5()* nicht automatisch unterstützt wird. Wenn der String mit einem anderen Salt-String verschlüsselt wird, werden auch dann unterschiedliche verschlüsselte Werte erzeugt, wenn zwei Benutzer das gleiche Passwort gewählt haben. In Beispiel 11-6 liefert die Verschlüsselung von `aardvark7` mit den beiden Salts `aa` und `bb` zwei ganz verschiedene Strings.

Ein übliches Verfahren ist es, die ersten zwei Zeichen des Benutzernamens als das Salt für *crypt()* zu verwenden. In der Regel führt das auch dann zu unterschiedlichen verschlüsselten Strings, wenn zwei Benutzer das gleiche Passwort gewählt haben, weil es unwahrscheinlich ist, dass auch die ersten beiden Zeichen der jeweiligen Benutzernamen gleich sind. Wenn Sie vorhaben, die *md5()*-Eingabe zu salzen, könnten Sie zusätzlich zum Passwort auch den Benutzernamen (oder einen Teil des Benutzernamens) an die *md5()*-Funktion übergeben, indem Sie einfach die beiden Strings verknüpfen.

Die Tabelle *users* wurde so angelegt, dass sie die 32 Zeichen langen Ergebnisse der *md5()*-Funktion speichern kann. Das folgende Codefragment zeigt, wie das Passwort mit der *md5()*-Funktion geschützt wird und der Tabelle *users* ein neuer Benutzer hinzugefügt wird:

```
function newUser($connection, $username, $password)
{
  // Die Prüfsumme für das Passwort erzeugen
  $stored_password = md5(trim($password));

  // Den neuen Benutzer einfügen
  $query = "INSERT INTO users SET password = '$stored_password',
                                  user_name = '$username'";

  if (!$result = @ mysql_query ($query, $connection))
    showerror( );
}
```

Die Funktion erwartet drei Parameter: eine MySQL-Datenbank-Verbindung, auf der die Datenbank *authentication* ausgewählt ist, sowie einen Benutzernamen und ein Passwort im Klartext. Im nächsten Abschnitt zeigen wir Ihnen, wie Sie einen Benutzer authentifizieren, indem Sie ein Passwort, das vom Benutzer angegeben wird, mit dem gespeicherten Passwort vergleichen. Noch später in diesem Kapitel zeigen wir Ihnen als einen Bestandteil eines vollständigen Authentifizierungs-Frameworks, wie Passwörter in der Tabelle *users* aktualisiert werden.

Weil *crypt()* und *md5()* Einweg-Verfahren sind, gibt es keine Möglichkeit, den ursprünglichen Wert zurückzuerhalten, nachdem ein Passwort gespeichert worden ist. Das verhindert manches wünschenswerte Feature. Beispielsweise ist es nicht möglich, einen Benutzer an sein Passwort zu erinnern, falls er es vergessen haben sollte. Das Wichtige aber ist, dass es – außer den ganz entschlossenen – so gut wie alle Versuche verhindert, Zugriff auf Passwörter zu erlangen.

Der Authentifizierungsvorgang

Wenn ein Skript einen Benutzernamen und ein Passwort authentifizieren muss, die auf eine Authentifizierungsaufforderung hin zurückgeliefert wurden, muss es diese Berechtigungen mit der Datenbank vergleichen. Dazu wird das vom Benutzer angegebene Passwort verschlüsselt. Dann wird eine Abfrage ausgeführt, um in der Tabelle *users* eine Zeile mit einem passenden Benutzernamen und einem verschlüsselten Passwort zu finden. Wenn eine Zeile gefunden wird, ist der Benutzer gültig.

Beispiel 11-7 zeigt die Funktion *authenticateUser()*, die Berechtigungen validiert. Die Funktion wird aufgerufen, indem ein Handle für einen verbundenen MySQL-Server, auf dem die Datenbank *authentication* ausgewählt ist, sowie der Benutzername und das Passwort übergeben werden, die aus der Authentifizierungsaufforderung gelesen wurden. Zunächst prüft die Funktion `$username` und `$password`. Wenn eine der Variablen nicht gesetzt ist, liefert die Funktion `false` zurück. Das Skript baut dann eine `SELECT`-Abfrage auf, um die *users*-Tabelle zu durchsuchen. Dabei werden `$username` und die Prüfsumme von `$password` verwendet, die mit der *md5()*-Funktion erzeugt wurde. Die Abfrage wird ausgeführt, und wenn eine Zeile gefunden wird, wurden `$username` und `$password` authentifiziert, und die Funktion liefert `true` zurück.

Beispiel 11-7: Authentifizierung eines Benutzers anhand eines verschlüsselten Passworts in der users-Tabelle

```
<?php

function authenticateUser($connection, $username, $password)
{
  // Die Parameter für den Benutzernamen und das Passwort prüfen
  if (!isset($username) || !isset($password))
    return false;

  // Eine Prüfsumme des Passworts erzeugen, das bei
  // der Authentifizierungsaufforderung erhalten wurde
  $password_digest = md5(trim($password));

  // Die SQL-Abfrage formulieren, mit der nach dem Benutzer gesucht wird
  $query = "SELECT password FROM users WHERE user_name = '{$username}'
            AND password = '{$password_digest}'";

  if (!$result = @ mysql_query ($query, $connection))
    showerror( );

  // Wurde genau eine Zeile gefunden? Dann wurde der Benutzer gefunden.
  if (mysql_num_rows($result) != 1)
    return false;
  else
    return true;
}
?>
```

Es ist davon auszugehen, dass die Funktion *authenticateUser()* von vielen Skripten verwendet wird. Deswegen ist es praktisch, sie in einer *require*-Datei zu speichern. Wenn der Code beispielsweise in der Datei *authentication.inc* gespeichert wäre, könnten wir Beispiel 11-2 so umschreiben, dass es die Datenbank-Authentifizierungsfunktion verwendet, indem es diese Datei lädt. Die veränderte Version wird in Beispiel 11-8 gezeigt.

Beispiel 11-8: Eine umgeschriebene Version von Beispiel 11-2, die eine Datenbank-Authentifizierung verwendet

```
<?php
require "authentication.inc";
require "db.inc";
require_once "HTML/Template/ITX.php";

$template = new HTML_Template_ITX("./templates");
$template->loadTemplatefile("example.11-3.tpl", true, true);

if (!($connection = mysql_connect("localhost", "lucy", "secret")))
    die("Could not connect to database");

if (!mysql_selectdb("authentication", $connection))
   showerror( );
```

Beispiel 11-8: Eine umgeschriebene Version von Beispiel 11-2, die eine Datenbank-Authentifizierung verwendet (Fortsetzung)

```
$username = mysqlclean($_SERVER, "PHP_AUTH_USER", 50, $connection);
$password = mysqlclean($_SERVER, "PHP_AUTH_PW", 32, $connection);

if (!authenticateUser($connection, $username, $password))
{
  // Keine Berechtigungen gefunden.
  // Antwort mit Authentifizierungsaufforderung versenden
  header("WWW-Authenticate: Basic realm=\"Flat Foot\"");
  header("HTTP/1.1 401 Unauthorized");

  // Den Body der Antwort einrichten, der angezeigt wird,
  // wenn der Benutzer die Authentifizierungsaufforderung abbricht
  $template->touchBlock("challenge");
  $template->show( );
  exit;
}
else
{
  // Den authentifizierten Benutzer begrüßen
  $template->touchBlock("authenticated");
  $template->show( );
}
?>
```

Andere Daten in einer Datenbank verschlüsseln

Die PHP-Funktionen *crypt()* und *md5()* können nur verwendet werden, um Passwörter, persönliche Identifikationskennnummern (PINs) usw. zu verschlüsseln. Es sind Einweg-Funktionen – nachdem das ursprüngliche Passwort einmal verschlüsselt und gespeichert worden ist, können Sie es nicht mehr zurückerhalten. (Wie bereits erwähnt, ist der Rückgabewert von *md5()* eine Signatur oder ein Fingerabdruck der Nachricht und keine verschlüsselte Kopie.) Auf diesem Grund können diese Funktionen nicht verwendet werden, um schützenswerte Informationen zu speichern, die eine Anwendung wieder abrufen muss. Beispielsweise können Sie sie nicht verwenden, um Kreditkartendaten zu speichern und abzurufen oder um ein geheimes Dokument zu verschlüsseln.

Um schützenswerte Informationen zu speichern, benötigen Sie Zwei-Weg-Funktionen, die einen geheimen Schlüssel verwenden, um die Daten zu verschlüsseln und zu entschlüsseln. Ein entscheidendes Problem bei der Verwendung eines Schlüssels zur Verschlüsselung und Entschlüsselung von Daten ist die Notwendigkeit, sicher mit dem Schlüssel umzugehen. Das Problem der Schlüsselverwaltung geht über den Horizont dieses Buchs hinaus. Trotzdem werden wir die Verschlüsselung im Abschnitt »Daten im Web schützen« kurz behandeln.

Wenn Sie Daten mit einer Zwei-Weg-Verschlüsselung speichern müssen, finden Sie in der Verschlüsselungsbibliothek `mcrypt` einen guten Werkzeugkasten. PHP bietet einen Satz von Funktionen für den Zugriff darauf. Um sie zu verwenden, müssen Sie die Biblio-

thek `libmcrypt` installieren und PHP dann mit dem Parameter `--with-mcrypt` kompilieren. Auf der PHP-Website finden Sie außerdem gebrauchsfertige Software für Microsoft Windows-Systeme. In diesem Buch werden wir die `mcrypt`-Bibliothek nicht behandeln. Aber unter *http://www.php.net/manual/en/ref.mcrypt.php* und *http://mcrypt.sourceforge.net/* können Sie weiter Informationen finden.

MySQL bietet außerdem die reversiblen Funktionen *encode()* und *decode()*, die in Kapitel 15 beschrieben werden.

Formularbasierte Authentifizierung

Bisher haben wir in diesem Kapitel Authentifizierungstechniken verwendet, die auf HTTP basieren. In diesem Abschnitt beschreiben wir, wie man Anwendungen aufbaut, die sich nicht auf die HTTP-Authentifizierung stützen, sondern stattdessen HTML-Formulare verwenden, um die Berechtigungen zu sammeln, und Sessions einsetzen, um ein Authentifizierungs-Framework zu implementieren. Wir erörtern, warum man es vorziehen könnte, die HTTP-Authentifizierung zu vermeiden, und betrachten die Typen von Anwendungen, bei denen die Authentifizierungsabwicklung über Formulare vorteilhaft ist.

Gründe dafür, die HTTP-Authentifizierung zu verwenden

Bevor Sie sich dazu entschließen, eine Anwendung aufzubauen, die ihre Authentifizierung selbst abwickelt, sollten Sie die Vorteile in Betracht ziehen, die Ihnen die Verwendung der HTTP-Authentifizierung bietet:

- Sie ist einfach zu verwenden. Um eine Anwendung zu schützen, müssen Sie bloß Ihren Webserver konfigurieren oder eine Datei anlegen.
- Der Ablauf der HTTP-Authentifizierung kann mit PHP-Code gesteuert werden, wenn eine Anwendung die Prüfung der Benutzerberechtigungen selbst in die Hand nehmen muss. Wie das geht, haben wir weiter oben in diesem Kapitel im Abschnitt »HTTP-Authentifizierung mit PHP verwalten« beschrieben.
- In Browser ist eine Unterstützung für das Aufnehmen und Speichern von Benutzerberechtigungen eingebaut.
- Die HTTP-Authentifizierung funktioniert gut bei zustandslosen Anwendungen.

Gründe dafür, die HTTP-Authentifizierung zu vermeiden

Einige Anwendungen, insbesondere sessionbasierte Anwendungen, die authentifizierte Benutzer nachhalten müssen, bringen Anforderungen mit sich, denen man mit der HTTP-Authentifizierung nur schwer genügen kann.

Browser erinnern sich an Passwörter

Benutzernamen und Passwörter, die in einen Authentifizierungsdialog eines Browsers (wie den, der in Abbildung 11-1 gezeigt wurde) eingegeben wurden, werden so lange aufbewahrt, bis das Browserprogramm beendet wird oder ein neuer Satz mit Berechtigungen aufgenommen wurde. Sie können einen Browser zwingen, Berechtigungen zu vergessen, indem Sie absichtlich mit Code antworten, der so tut, als sei der Benutzer nicht authentifiziert worden, auch wenn ein HTTP-Request authentifizierte Berechtigungen enthält. Das folgende Fragment tut genau das:

```
// Den Browser mit einer erneuten Authentifizierungs-
// aufforderung zum Vergessen zwingen ...
header("WWW-Authenticate: Basic realm=\"Flat Foot\"");
header("HTTP/1.1 401 Unauthorized");
```

Wenn ein Benutzer jedoch vergisst, sich auszuloggen – und wenn die Seite, die das Header-Feld `WWW-Authenticate` sendet, nicht erneut angefordert wird –, dann wird der allein gelassene Browser zu einem Sicherheitsrisiko. Durch Eingabe einer URL oder einen simplen Klick auf den ZURÜCK-Button kann sich ein anderer Benutzer Zugriff auf die Anwendung verschaffen, ohne erneut zur Authentifizierung aufgefordert zu werden.

Beschränkung auf den Authentifizierungsdialog des Browsers

Nutzt eine Anwendung die HTTP-Authentifizierung, gibt es nur die Möglichkeit, die Benutzerberechtigungen über das Dialogfenster aufzunehmen, das der Browser bietet. Vielleicht möchte eine Online-Anwendung aber eine Login-Seite in einem Stil präsentieren, der dem der Anwendung entspricht. Dazu könnte z.B. ein Template verwendet werden. Oder vielleicht soll die Login-Seite auch in einer anderen Sprache verfasst sein.

HTTP bietet keine Unterstützung für mehrere Bereiche

Bei einigen Anwendungen sind mehrere Logins erforderlich. Beispielsweise könnte es bei einem unternehmensweiten Informationssystem erforderlich sein, dass sich alle Benutzer einloggen müssen, um einfachen Zugriff zu erhalten. Für den Zugriff auf einen eingeschränkten Bereich der Site könnten dann ein zusätzlicher Benutzername und ein entsprechendes Passwort erforderlich sein. Aber mehrere `Authorization`-Header-Felder sind bei HTTP nicht zulässig.

Authentifizierung und sessionbasierte Anwendungen

In Kapitel 10 haben wir das Session-Management als Technik für den Aufbau zustandsbehafteter Anwendungen vorgestellt. Bei vielen Anwendungen, bei denen eine Authentifizierung erforderlich ist, wird eine Session angelegt, wenn sich ein Benutzer einloggt. Mit ihr wird die Interaktion nachgehalten, bis sich der Benutzer ausloggt oder die Session abläuft. Dieses Muster haben wir in Kapitel 10 eingeführt.

Das grundlegende Muster bei der sessionbasierten Authentifizierung ist, die Benutzerberechtigungen einmal zu authentifizieren und dann eine Session einzurichten, die diesen

Authentifizierungsstatus in Session-Variablen festhält. Berechtigungen werden über ein Formular aufgenommen und über ein Set-up-Skript vorverarbeitet. Das unterscheidet sich von der HTTP-Authentifizierung, bei der die Berechtigungen bei jeder Anfrage mit übermittelt werden. Wenn die Session abläuft (oder der Benutzer die Session zerstört), wird der Authentifizierungsstatus zerstört. Deswegen kann das Session-ID-Cookie, im Unterschied zu den authentifizierten HTTP-Berechtigungen, nach Ablauf der Session nicht mehr verwendet werden. Dadurch wird die Anwendung sicherer.

Es hat zwei Nachteile, wenn Benutzerberechtigungen über ein Formular aufgenommen werden und der Authentifizierungsstatus in einer Session gespeichert wird. Erstens werden der Benutzername und das Passwort nicht verschlüsselt, wenn sie vom Browser an den Server übermittelt werden. Deswegen werden in den PHP-Beispielen, die wir im Rest dieses Kapitels vorstellen, der Benutzername und das Passwort im Klartext übertragen. Das Problem lässt sich mit dem SSL-Protokoll lösen, das wir später in diesem Kapitel besprechen werden. Zweitens können Sessions gekapert werden, weil der Status der Session verwendet wird, um den Zugriff auf die Anwendung zu kontrollieren. Mit dem Kapern von Sessions werden wir uns jetzt befassen.

Kapern von Sessions

Werden Session-Variablen verwendet, um den Authentifizierungsstatus festzuhalten, kann eine Anwendung gekapert werden. Wenn eine Anfrage an eine sessionbasierte Anwendung geschickt wird, schließt der Browser in sie (üblicherweise als ein Cookie) den Session-Identifier ein, um auf die authentifizierte Session zuzugreifen. Anstatt nach Benutzernamen und Passwörtern zu schnüffeln, kann ein Hacker eine Session-ID verwenden, um eine vorhandene Session zu kapern.

Denken Sie an eine Online-Banking-Anwendung, bei der der Hacker darauf wartet, dass sich ein echter Benutzer einloggt. Dann schließt der Hacker die Session-ID in eine Anfrage ein und überweist Geldbeträge auf sein eigenes Konto. Wenn die Session nicht verschlüsselt ist, ist es kein Problem, die Session-ID zu lesen. Deswegen empfehlen wir, Anwendungen, die Benutzernamen, Passwörter, Sessions identifizierende Cookies oder persönliche Daten übermitteln, durch Verschlüsselung zu schützen.

Selbst wenn die Verbindung verschlüsselt ist, kann die Session-ID immer noch gefährdet sein. Wenn die Session-ID in einem Cookie auf dem Client gespeichert ist, kann man den Browser so austricksen, dass er das Cookie unverschlüsselt versendet. Das kann passieren, wenn das Cookie vom Server ohne den Parameter `secure` eingerichtet wurde, der verhindert, dass Cookies über eine unsichere Verbindung übermittelt werden. Wie Sie das PHP-Session-Management so einrichten, dass Cookies gesichert werden, wird in Kapitel 10 behandelt.

Versuche, Sessions zu kapern, können auch weniger trickreich sein. Ein Hacker kann eine Session kapern, indem er zufällig Session-IDs ausprobiert und darauf hofft, auf eine vorhandene Session zu stoßen. Auf einer sehr aktiven Site können zu einem Zeitpunkt viele Tausende Sessions existieren. Das erhöht die Erfolgsaussichten eines solchen Versuchs. Eine Vorsichtsmaßnahme ist, die Anzahl der inaktiven Sessions zu reduzieren, indem

man, wie in Kapitel 10 erläutert wurde, die maximale Lebensdauer inaktiver Sessions klein hält.

Versuche, Sessions zu kapern, durch Aufzeichnen der IP-Adressen feststellen

Weiter oben in diesem Kapitel haben wir gezeigt, wie man auf die IP-Adresse des Browsers zugreift, wenn man einen Request verarbeitet. Das in Beispiel 11-4 gezeigte Skript prüft die IP-Adresse, die in der Variablen `$_SERVER["REMOTE_ADDR"]` gesetzt ist, anhand eines hartcodierten Strings, der den Zugriff auf Benutzer einschränkt, die sich in einem bestimmten Subnetz befinden.

Die IP-Adresse des Clients kann auch verwendet werden, um dem Kapern von Sessions vorzubeugen. Falls die in der Variablen `$_SERVER["REMOTE_ADDR"]` gesetzte IP-Adresse als Session-Variable gespeichert wird, wenn ein Benutzer zum ersten Mal eine Verbindung mit einer Anwendung herstellt, können nachfolgende Anfragen geprüft und nur dann zugelassen werden, wenn sie von derselben IP-Adresse ausgingen. Wie das geht, zeigen wir Ihnen im nächsten Abschnitt.

Bei der Verwendung von IP-Adressen, die aus einem HTTP-Request aufgezeichnet wurden, gibt es aber Einschränkungen. Oft konfigurieren Netzwerk-Administratoren Proxy-Server so, dass die ursprüngliche IP-Adresse verborgen und durch die Adresse des Proxy-Servers ersetzt wird. Alle Benutzer, die sich über einen solchen Proxy-Server mit einer Anwendung verbinden, scheinen auf demselben Rechner zu arbeiten. Einige große Sites – wie die einer großen Universität – können sogar mit mehreren Proxy-Servern arbeiten, um die Serverlast zu verteilen. Deswegen kann es sein, dass es so aussieht, als ob die IP-Adresse wechselt, obwohl aufeinander folgende Anfragen von demselben Benutzer kommen.

Ein sessionbasiertes Authentifizierungs-Framework

Das in diesem Abschnitt entwickelte Authentifizierungs-Framework entspricht dem Muster, das in Kapitel 10 beschrieben wurde, und beruht auf Techniken, die wir weiter oben in diesem Kapitel entwickelt haben. In diesem Abschnitt

- entwickeln wir ein Login-Skript, das ein Formular verwendet, um die Benutzerberechtigungen entgegenzunehmen,
- authentifizieren wir die Benutzerberechtigungen anhand der geschützten Passwörter, die in der Tabelle *users* gespeichert wurden,
- zeigen wir, wie Session-Variablen eingerichtet werden, um eine Session-Authentifizierung zu unterstützen und Versuche zu bemerken, Sessions zu kapern,
- entwickeln wir die *sessionAuthenticate()*-Funktion, die jede Seite schützt, für die eine Authentifizierung erforderlich ist,
- entwickeln wir eine Logout-Funktion, die eine Session zerstört, und
- entwickeln wir ein Skript, das es dem Benutzer ermöglicht, sein Passwort zu ändern.

Die in diesem Abschnitt vorgestellten Skripten wurden so einfach gehalten wie möglich, um die Grundlagen zu erläutern. Sie verwenden die Datenbank *authentication* und die Tabelle *users*, die weiter oben in diesem Kapitel beschrieben wurden. Die Datenbank-Verbindung wird über die Benutzerin `lucy` und das Passwort `secret` hergestellt. Ein komplexeres Authentifizierungs-Framework, das auf den Skripten basiert, die hier beschrieben werden, wird bei unserer Online-Weinhandlung in den Kapiteln 16 bis 20 vorgestellt.

Ein Überblick über den Code

Das grundlegende Muster sessionbasierter Authentifizierung sieht vor, die Benutzerberechtigungen einmal zu authentifizieren und dann eine Session einzurichten, die diesen Authentifizierungsstatus als Session-Variable aufzeichnet. Berechtigungen werden mit der Seite *login.html* aufgenommen, die in Beispiel 11-9 gezeigt wird, und vom Skript *logincheck.php* verarbeitet, das in Beispiel 11-10 zu sehen ist.

Anwendungsskripten – wie das Skript *home.php* in Beispiel 11-12 – prüfen zunächst den Status der Session-Variablen für die Authentifizierung, bevor irgendwelcher anderer Code ausgeführt wird. Diese Prüfung erfolgt mit der Funktion *sessionAuthenticate()*. Wenn die Prüfung fehlschlägt, wird der Benutzer zum Skript *logout.php* umgeleitet, das in Beispiel 11-14 gezeigt wird. Mit ihm wird die Session explizit zerstört. Das Skript *logout.php* kann auch direkt aufgerufen werden, und wie in *home.php* wird üblicherweise in den meisten Seiten der Anwendung ein Link darauf eingeschlossen.

Die Funktionen, die innerhalb des Frameworks wiederverwendet werden, werden in der *require*-Datei *authentication.inc* implementiert, die in Beispiel 11-11 zu sehen ist. Die Datei enthält die Funktionen *authenticateUser()*, mit der die vom Benutzer angegebenen Berechtigungen mit denen in der Datenbank verglichen werden (diese Funktion wird in Beispiel 11-7 gezeigt), und *sessionAuthenticate()*.

Das Modul für die Passwortänderung wird in Beispiel 11-16 und Beispiel 11-18 gezeigt. Beispiel 11-16 enthält das Skript *password.php*, das das Formular für die Passwortänderung anzeigt, um das aktuelle Passwort und ein neues Passwort aufzunehmen. Beispiel 11-18 präsentiert das Skript *changepassword.php*, das die Benutzerdaten validiert und, falls die Validierung erfolgreich ist, das Passwort ändert. Bei Erfolg oder bei einem Fehlschlag leitet das Skript *changepassword.php* zur Seite für die Passwortänderung um und zeigt eine Nachricht an, die den Benutzer informiert.

Die Login-Seite

Beispiel 11-9 zeigt die Seite *login.html* mit einem Formular, das den Benutzernamen und das Passwort aufnimmt. Die Login-Seite enthält keinerlei PHP-Code.

Beispiel 11-9: Die Login-Seite

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html401/loose.dtd">
<html>
```

Beispiel 11-9: Die Login-Seite (Fortsetzung)

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Login</title>
</head>
<body>
<h1>Application Login Page</h1>
<form method="POST" action="logincheck.php">
<table>
  <tr>
    <td>Enter your username:</td>
    <td><input type="text" size="10" name="loginUsername"></td>
  </tr>
  <tr>
    <td>Enter your password:</td>
    <td><input type="password" size="10" name="loginPassword"></td>
  </tr>
</table>
<p><input type="submit" value="Log in">
</form>
</body>
</html>
```

Proxy-Server, Web-Gateways und Webserver loggen häufig die URLs, die angefragt werden. Deswegen versendet die Seite die Formular-Eingabefelder mit der POST-Methode anstelle der GET-Methode, bei der die Feldwerte in der URL codiert werden. Das verhindert, dass Benutzerberechtigungen in Log-Dateien erscheinen.

Das Setup-Skript

Das in Beispiel 11-10 gezeigte Skript *logincheck.php* authentifiziert den Benutzer, indem die POST-Variablen verarbeitet werden, die auf der Seite *login.html* gesammelt wurden, und richtet die Session-Variablen ein, die den Authentifizierungsstatus aufzeichnen. Dieses Skript erzeugt keine Ausgaben außer einem Location-Header, mit dem zur Homepage der Anwendung oder, falls die Authentifizierung scheitert, zur Logout-Seite umgeleitet wird.

Beispiel 11-10: Das Setup-Skript

```
<?php
require 'authentication.inc';
require 'db.inc';

if (!$connection = @ mysql_connect("localhost", "lucy", "secret"))
  die("Cannot connect");

// Die Daten säubern, die im Formular gesammelt wurden
$loginUsername = mysqlclean($_POST, "loginUsername", 10, $connection);
$loginPassword = mysqlclean($_POST, "loginPassword", 10, $connection);
```

Beispiel 11-10: Das Setup-Skript (Fortsetzung)

```
if (!mysql_selectdb("authentication", $connection))
  showerror( );

session_start( );

// Den Benutzer authentifizieren
if (authenticateUser($connection, $loginUsername, $loginPassword))
{
  // Den Login-Benutzernamen loginUsername registrieren
  $_SESSION["loginUsername"] = $loginUsername;

  // Die IP-Adresse speichern, die die Session gestartet hat
  $_SESSION["loginIP"] = $_SERVER["REMOTE_ADDR"];

  // Zur ersten Seite der Anwendung umleiten
  header("Location: home.php");
  exit;
}
else
{
  // Die Authentifizierung ist gescheitert:  Eine Logout-Nachricht erstellen
  $_SESSION["message"] =
    "Could not connect to the application as '{$loginUsername}'";

  // Zur Logout-Seite umleiten
  header("Location: logout.php");
  exit;
}
?>
```

Der Benutzername und das Passwort werden aus dem superglobalen Array `$_POST` gelesen und gesäubert. Dann werden der Benutzername und das Passwort der Funktion *authenticateUser()* übergeben. Wenn die Funktion *authenticateUser()* `true` zurückliefert, wurde der Benutzer erfolgreich authentifiziert. Dann richtet das Skript die Session-Variablen `$_SESSION["loginUsername"]` und `$_SESSION["loginIP"]` ein. Schließlich wird der `Location`-Header versendet, um den Browser zum Skript *home.php* umzuleiten. Wenn die Benutzerberechtigungen nicht authentifiziert werden, richtet das Skript die Session-Variable `message` ein und leitet zum Skript *logout.php* um.

Die require-Datei authentication.inc

Alle Seiten, die vom Authentifizierungs-Framework geschützt werden, müssen die Session-Variablen `$_SESSION["loginUsername"]` und `$_SESSION["loginIP"]` prüfen, um vor der Ausführung anderen Codes sicherzustellen, dass der Benutzer erfolgreich authentifiziert wurde. Die in Beispiel 11-11 gezeigte Funktion *sessionAuthenticate()* führt diese Prüfung durch. Sie wird in die Datei *authentication.inc* eingeschlossen.

Beispiel 11-11: Die Funktionen sessionAuthenticate() und authenticateUser( )

```
<?php

function authenticateUser($connection, $username, $password)
{
  // Die Parameter für Benutzername und Passwort prüfen
  if (!isset($username) || !isset($password))
    return false;

  // Eine Prüfsumme des Passworts erstellen, das auf die
  // Autorisierungsaufforderung erhalten wurde
  $password_digest = md5(trim($password));

  // Die SQL-Abfrage formulieren, mit der der Benutzer gesucht wird
  $query = "SELECT password FROM users WHERE user_name = '{$username}'
            AND password = '{$password_digest}'";

  // Die Abfrage ausführen
  if (!$result = @ mysql_query ($query, $connection))
    showerror( );

  // Wurde genau eine Zeile gefunden? Dann wurde der Benutzer gefunden
  if (mysql_num_rows($result) != 1)
    return false;
  else
    return true;
}

// Mit einer Session verbinden und prüfen, ob der Benutzer
// authentifiziert wurde und ob die IP-Adresse der Aufrufers
// der Adresse entspricht, über die die Session gestartet wurde.
function sessionAuthenticate( )
{
  // Prüfen, ob der Benutzer noch nicht eingeloggt ist
  if (!isset($_SESSION["loginUsername"]))
  {
    // Die Anfrage identifiziert keine Session
    $_SESSION["message"] = "You are not authorized to access the URL
                            {$_SERVER["REQUEST_URI"]}";

    header("Location: logout.php");
    exit;
  }

  // Prüfen, ob die Anfrage von einer anderen IP-Adresse kommt als zuvor
  if (!isset($_SESSION["loginIP"]) ||
     ($_SESSION["loginIP"] != $_SERVER["REMOTE_ADDR"]))
  {
    // Die Anfrage stammt nicht von dem Rechner, der verwendet wurde,
    // um die Session zu starten.
    // WAHRSCHEINLICH EIN VERSUCH, EINE SESSION ZU KAPERN
```

Beispiel 11-11: Die Funktionen sessionAuthenticate() und authenticateUser() (Fortsetzung)

```
    $_SESSION["message"] = "You are not authorized to access the URL
                            {$_SERVER["REQUEST_URI"]} from the address
                            {$_SERVER["REMOTE_ADDR"]}";

    header("Location: logout.php");
    exit;
  }
}

?>
```

Die Funktion *sessionAuthenticate()* führt zwei Tests durch. Zunächst prüft sie, ob die Session-Variable `$_SESSION["loginUsername"]` nicht gesetzt ist. Dann ist der Benutzer nicht eingeloggt. Dann prüft sie, ob die Session-Variable `$_SESSION["loginIP"]` nicht gesetzt ist oder ob sie nicht den gleichen Wert hat wie die IP-Adresse des Clients, der die aktuelle Anfrage ausgeführt hat. Dann handelt es sich wahrscheinlich um einen Versuch, die Session zu kapern. Wenn einer der Tests fehlschlägt, wird eine `$_SESSION["message"]`-Variable mit einer passenden Nachricht eingerichtet. Dann wird ein `Location`-Header-Feld verwendet, um den Browser zum Logout-Skript umzuleiten.

Beispiel 11-11 enthält außerdem die Funktion *authenticateUser()*, die bereits in Beispiel 11-7 vorgestellt wurde.

Anwendungsskripten und -seiten

Beispiel 11-12 zeigt das Skript *home.php*, das die Datei *authentication.inc* und die Funktion *sessionAuthenticate()* verwendet. Wenn der Benutzer diese Seite anfordert, bevor er eingeloggt ist, wird er zur Seite *logout.php* umgeleitet. Wenn er eingeloggt ist, wird die Seite *home.php* angezeigt.

Beispiel 11-12: Die Homepage der Anwendung

```
<?php
require "authentication.inc";
require_once "HTML/Template/ITX.php";

session_start( );

// Mit einer authentifizierten Session verbinden oder zu logout.php umleiten
session_authenticate( );

$template = new HTML_Template_ITX("./templates");
$template->loadTemplatefile("home.tpl", true, true);

$template->setVariable("USERNAME", $_SESSION["loginUsername"]);
$template->parseCurrentBlock( );
$template->show( );
?>
```

Das Skript verwendet das Template *home.tpl*, das in Beispiel 11-13 zu sehen ist, um die Variable `$_SESSION["loginUsername"]` anzuzeigen, die angibt, wer eingeloggt ist. Dieses Skript bietet Links zum Ausloggen und um das Passwort des Benutzers zu ändern.

Beispiel 11-13: Das Template home.tpl, das mit Beispiel 11-12 verwendet wird

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html401/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Home</title>
</head>
<body>
  <h1>Welcome to the application</h1>
  You are logged on as {USERNAME}
  <p><a href="password.php">Change Password</a>
  <p><a href="logout.php">Logout</a>
</body>
</html>
```

Das Logout-Skript

Das Skript *logout.php* wird in Beispiel 11-14 gezeigt. Es wird entweder von einem anderen Skript (wie *logincheck.php*) aufgerufen, wenn der Benutzer den Authentifizierungsvorgang nicht erfolgreich abschließt, oder kann von einem Benutzer (beispielsweise über die im vorigen Abschnitt gezeigte Seite *home.php*) aufgerufen werden, um die Session explizit zu beenden.

Beispiel 11-14: Das Logout-Skript

```
<?php
  require_once "HTML/Template/ITX.php";
  session_start();

  $message = "";

  // Ein authentifizierter Benutzer hat sich ausgeloggt -- wir sind nett
  // und bedanken uns, dass er unsere Anwendung verwendet hat
  if (isset($_SESSION["loginUsername"]))
    $message .= "Thanks {$_SESSION["loginUsername"]} for
                 using the Application.";

  // Irgendein Skript, wahrscheinlich das Setup-Skript, könnte eine
  // Logout-Nachricht angelegt haben
  if (isset($_SESSION["message"]))
  {
    $message .= $_SESSION["message"];
    unset($_SESSION["message"]);
  }
```

Beispiel 11-14: Das Logout-Skript (Fortsetzung)

```
  // Die Session zerstören
  session_destroy( );

  // Die Seite (einschließlich der Nachricht) anzeigen
  $template = new HTML_Template_ITX("./templates");
  $template->loadTemplatefile("logout.tpl", true, true);
  $template->setVariable("MESSAGE", $message);
  $template->parseCurrentBlock( );
  $template->show( );
?>
```

In *logout.php* gibt es keinen Aufruf der Funktion *sessionAuthenticate()*, mit dem geprüft würde, ob der Benutzer authentifiziert ist. Deswegen müssen wir die Datei *authentication.inc* nicht in das Skript laden. Stattdessen ruft *logout.php* die Funktion *session_start()* auf und prüft, ob eine der Session-Variablen `$_SESSION["loginUsername"]` und `$_SESSION ["message"]` gesetzt ist. Wenn eine davon gesetzt ist, wird sie verwendet, um eine Nachricht zu erzeugen, die dem Benutzer angezeigt wird:

- Die Variable `$_SESSION["message"]` wird in den Skripten *logincheck.php* oder *authentication.inc* erzeugt, wenn die Berechtigungen nicht authentifiziert werden können. Sie wird verwendet, um zu erläutern, warum der Vorgang fehlgeschlagen ist.
- Die Variable `$_SESSION["loginUsername"]` wird in *logout.php* genutzt, um dem Benutzer zu danken, dass er die Anwendung verwendet hat.

Ist die Nachricht vollständig, zerstört das Skript die Session, indem es die Funktion *session_destroy()* aufruft. Die Logout-Seite gibt die Variable `$message` über das Template *logout.tpl* aus, das in Beispiel 11-15 zu sehen ist. Diese Seite bietet auch einen Link zurück zur Seite *login.html*.

Beispiel 11-15: Das Template logout.tpl, das mit Beispiel 11-14 verwendet wird

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html401/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Logout</title>
</head>
<body>
  <h1>Application Logout Page</h1>
  {MESSAGE}
  <p>Click <a href="login.html">here</a> to log in.
</body>
</html>
```

Password-Management

Die Skripten *password.php* in Beispiel 11-16 und *changepassword.php* in Beispiel 11-18 ermöglichen es dem Benutzer, sein Passwort zu ändern. Beide Skripten laden zunächst die Datei *authentication.inc* und rufen die Funktion *sessionAuthenticate()* auf und erlauben nur dann einen Zugriff, wenn der Benutzer erfolgreich authentifiziert wurde.

Beispiel 11-16: Das Skript password.php mit dem Formular zur Änderung des Passworts

```
<?php
require "authentication.inc";
require_once "HTML/Template/ITX.php";

session_start( );

// Mit einer authentifizierten Session verbinden oder zu logout.php umleiten
sessionAuthenticate( );

$message = "";

// Prüfen, ob es eine Passwort-Fehlermeldung gibt
if (isset($_SESSION["passwordMessage"]))
{
  $message = $_SESSION["passwordMessage"];
  unset($_SESSION["passwordMessage"]);
}

// Die Seite (einschließlich der Nachricht) anzeigen
$template = new HTML_Template_ITX("./templates");
$template->loadTemplatefile("password.tpl", true, true);
$template->setVariable("USERNAME", $_SESSION["loginUsername"]);
$template->setVariable("MESSAGE", $message);
$template->parseCurrentBlock( );
$template->show( );
?>
```

Das Skript *password.php* zeigt ein Formular an, in das das ursprüngliche Passwort und zweimal das neue Passwort eingegeben wird. Das neue Passwort soll zweimal eingegeben werden, um das Risiko zu vermindern, dass das Passwort durch einen Tippfehler unbrauchbar wird. Das Skript nutzt das in Beispiel 11-17 gezeigte Template *password.tpl*. Darin gibt es zwei Template-Platzhalter: `USERNAME` wird verwendet, um den Namen des eingeloggten Benutzers anzuzeigen, und `MESSAGE`, um eine Nachricht anzuzeigen, die in einer Session-Variable gespeichert ist, die von *changepassword.php* gesetzt wird. Nachdem eine gespeicherte Nachricht angezeigt worden ist, wird die Variable aus dem Session-Speicher gelöscht, damit die Nachricht nicht noch einmal erscheint.

Beispiel 11-17: Das Template password.tpl für Beispiel 11-16

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html401/loose.dtd">
<html>
```

Beispiel 11-17: Das Template password.tpl für Beispiel 11-16 (Fortsetzung)

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Password Change</title>
</head>
<body>
  <h1>Change Password for {USERNAME}</h1>
  {MESSAGE}
  <form method="POST" action="changepassword.php">
  <table>
    <tr>
      <td>Enter your existing password:</td>
      <td><input type="password" size="10" name="oldPassword"></td>
    </tr>
    <tr>
      <td>Enter your new password:</td>
      <td><input type="password" size="10" name="newPassword1"></td>
    </tr>
    <tr>
      <td>Re-enter your new password:</td>
      <td><input type="password" size="10" name="newPassword2"></td>
    </tr>
  </table>
  <p><input type="submit" value="Update Password">
  </form>
  <p><a href="home.php">Home</a>
  <p><a href="logout.php">Logout</a>
</body>
</html>
```

Die Daten, die in das Passwort-Formular eingegeben werden, werden vom Skript *changepassword.php* in Beispiel 11-18 verarbeitet.

Beispiel 11-18: Das Skript changepassword.php

```
<?php
require "authentication.inc";
require "db.inc";

session_start();

// Mit einer authentifizierten Session verbinden oder zu logout.php umleiten
sessionAuthenticate();

if (!$connection = @ mysql_connect("localhost", "lucy", "secret"))
  die("Cannot connect");

// Die vom Benutzer eingegebenen Daten säubern
$oldPassword = mysqlclean($_POST, "oldPassword", 10, $connection);
$newPassword1 = mysqlclean($_POST, "newPassword1", 10, $connection);
$newPassword2 = mysqlclean($_POST, "newPassword2", 10, $connection);
```

Beispiel 11-18: Das Skript changepassword.php (Fortsetzung)

```
if (!mysql_selectdb("authentication", $connection))
  showerror( );

if (strcmp($newPassword1, $newPassword2) == 0 &&
  authenticateUser($connection, $_SESSION["loginUsername"], $oldPassword))
{
  // Das Passwort des Benutzers darf aktualisiert werden

  // Erzeuge die Prüfsumme des Passworts
  $digest = md5(trim($newPassword1));

  // Die Zeile für den Benutzer aktualisieren
  $update_query = "UPDATE users SET password = '{$digest}'
                   WHERE user_name = '{$_SESSION["loginUsername"]}'";

  if (!$result = @ mysql_query ($update_query, $connection))
    showerror( );

  $_SESSION["passwordMessage"] =
    "Password changed for '{$_SESSION["loginUsername"]}'";
}
else
{
  $_SESSION["passwordMessage"] =
    "Could not change password for '{$_SESSION["loginUsername"]}'";
}

// Zum Passwort-Formular umleiten
header("Location: password.php");
?>
```

Die Felder `oldPassword`, `newPassword1` und `newPassword2` werden aus dem superglobalen Array `$_POST` gelesen und mit der Funktion *mysqlclean()* gesäubert. Wenn die beiden Felder mit dem neuen Passwort den gleichen Wert enthalten und das aktuelle Passwort für den aktuell eingeloggten Benutzer gültig ist, wird der Aktualisierungscode ausgeführt. Wie bereits erwähnt wurde, sorgt die doppelte Erfassung des neuen Passworts dafür, dass Tippfehler abgefangen werden. Der Aufruf der Funktion *authenticateUser()* stellt sicher, dass nur der Benutzer selbst das Passwort ändern kann.

Nachdem die Felddaten geprüft wurden, kann das Passwort in der Datenbank aktualisiert werden. Die Zeile für den Benutzer wird jetzt mit der MD5-Prüfsumme des neuen Passworts aktualisiert, und die Variable `$_SESSION["passwordMessage"]` wird gesetzt, um anzuzeigen, dass das Passwort geändert worden ist. Diese Nachricht wird dann vom Skript *password.php* angezeigt.

Wenn die Prüfung der Felddaten nicht erfolgreich ist – wenn also die zwei Eingaben des neuen Passworts nicht übereinstimmen oder das aktuelle Passwort nicht gültig ist –, wird die Variable `$_SESSION["passwordMessage"]` gesetzt, um anzuzeigen, dass das Passwort nicht geändert werden konnte.

Das Skript *changepassword.php* erzeugt keine Ausgaben, setzt aber das Header-Feld `Location`, um den Browser zur Seite *password.php* umzuleiten.

Daten im Web schützen

Das Web ist keine sichere Umgebung. Die offene Natur des Netzwerks und der Webprotokolle TCP, IP und HTTP hat die Entwicklung vieler Tools ermöglicht, die Daten abhören können, die zwischen Webbrowsern und -servern übermittelt werden. Es ist möglich, den vorübergehenden Netzwerkverkehr zu belauschen und den Inhalt von HTTP-Requests und -Responses zu lesen. Gibt ein Hacker sich etwas mehr Mühe, kann er den Netzwerkverkehr manipulieren und sich sogar als anderer Benutzer ausgeben.

Wenn eine Anwendung schützenswerte Informationen über das Web übermittelt, sollte eine verschlüsselte Verbindung zwischen Webbrowser und -server verwendet werden. Eine verschlüsselte Verbindung ist in folgenden Fällen wünschenswert:

- Wenn auf dem Server schützenswerte Informationen gespeichert sind wie geheime Unternehmensdokumente oder Kontostände von Bankkonten.
- Wenn Benutzerberechtigungen verwendet werden, um auf schützenswerte Dienste wie Online-Banking oder die Administration einer Anwendung zuzugreifen.
- Wenn vom Benutzer persönliche Informationen wie Kreditkartennummern entgegengenommen werden.
- Wenn vom Server Session-IDs verwendet werden, um HTTP-Requests mit Session-Variablen zu verknüpfen und die Session gegen Kaperversuche abgesichert werden muss.

Auch wenn auf Ihre Anwendung keine dieser Bedingungen zutrifft, ist es manchmal nicht schlecht, bei kommerziellen Anwendungen eine Verschlüsselung einzusetzen. Wenn man wegen der Kompromittierung eines Systems eine schlechte Presse bekommt, ist es eigentlich egal, ob dabei private oder öffentliche Daten betroffen waren. Der Ruf des Unternehmens leidet darunter auf jeden Fall.

In diesem Abschnitt befassen wir uns damit, wie man mit SSL verschlüsselte Daten über das Web verschickt. Dabei konzentrieren wir uns auf die grundlegende Funktionsweise von SSL. Eine Installations- und Konfigurationsanleitung für SSL und den Apache-Webserver auf Unix- und Mac OS X-Plattformen ist in den Anhängen A bis C enthalten. Auch unter Microsoft Windows ist es möglich, einen sicheren Webserver einzurichten. Damit werden wir uns in diesem Buch aber nicht befassen.

Dieser Abschnitt soll das Thema Verschlüsselung nicht abschließend behandeln. Wir beschränken unsere kurze Betrachtung auf die Features von SSL und darauf, wie SSL den Webverkehr absichern kann. Ausführlichere Informationen zu kryptographischen Systemen finden Sie in den Referenzen, die in Anhang G aufgeführt werden.

Das Secure Sockets Layer-Protokoll

Die Daten, die zwischen Webservern und Browsern übertragen werden, können mit den Verschlüsselungsdiensten des Secure Socket Layer-Protokolls (SLL) geschützt werden. Das SSL-Protokoll hat drei Ziele:

Privatsphäre oder Vertraulichkeit
: Der Inhalt einer Nachricht, die über das Internet übermittelt wird, wird vor Beobachtern geschützt.

Unversehrtheit
: Die Inhalte der empfangenen Nachricht sind vollständig und wurden nicht manipuliert.

Authentizität
: Der Sender und der Empfänger einer Nachricht können beide sicher bezüglich der Identität des jeweils anderen sein.

SSL wurde ursprünglich von Netscape entwickelt, und es gibt zwei Versionen: SSL v2.0 und SSL v3.0. Wir werden die Unterschiede hier nicht erläutern, aber Version 3.0 unterstützt mehr Sicherheitsfunktionen als 2.0. Das SSL-Protokoll ist eigentlich kein Standard, und die Internet Engineering Task Force (IETF) hat als Ersatz für SSL v3.0 das *Transport Layer Security 1.0*-Protokoll (TLS-Protokoll) vorgeschlagen. Zurzeit sind SSL v3.0 und TLS fast identisch. Mehr Informationen zu TLS finden Sie unter *http://ietf.org/rfc/rfc2246.txt?number=2246*.

Die SSL-Architektur

Wenn Sie verstehen wollen, wie SSL funktioniert, müssen Sie verstanden haben, wie Browser und Webserver HTTP-Nachrichten verschicken und empfangen.

Browser verschicken HTTP-Requests, indem sie die TCP/IP-Netzwerk-Software ihres Systems aufrufen, die sich darum kümmert, Internetdaten zu empfangen und zu verschicken. Wenn eine Anfrage abgeschickt werden soll (beispielsweise, weil ein Benutzer auf einen Link geklickt hat), formuliert der Browser den HTTP-Request und nutzt den TCP/IP-Netzwerkdienst des Systems, um die Anfrage an den Server zu schicken. TCP/IP kümmert sich nicht darum, dass es sich um eine HTTP-Nachricht handelt. Es ist nur dafür verantwortlich, dass die vollständige Nachricht ihr Ziel erreicht. Wenn ein Webserver eine Nachricht empfängt, liest er die Daten aus dem TCP/IP-Dienst seines Systems und interpretiert sie dann als HTTP-Nachricht. Das Verhältnis zwischen HTTP und TCP/IP betrachten wir in Anhang D ausführlicher.

Wie in Abbildung 11-3 gezeigt, operiert das SSL-Protokoll als eine Schicht zwischen dem Browser und den TCP/IP-Diensten des Systems. Ein Browser übergibt die HTTP-Nachricht an die SSL-Schicht, damit sie verschlüsselt wird, bevor sie an den TCP/IP-Dienst des Systems übergeben wird. Die SSL-Schicht des Webserver-Systems entschlüsselt die Nachricht aus dem TCP/IP-Dienst und gibt sie dann an den Webserver weiter. Ist SSL einmal installiert und der Webserver richtig konfiguriert, werden die HTTP-Requests

und -Responses automatisch verschlüsselt. Zur Nutzung des SSL-Diensts sind keine PHP-Skripten erforderlich.

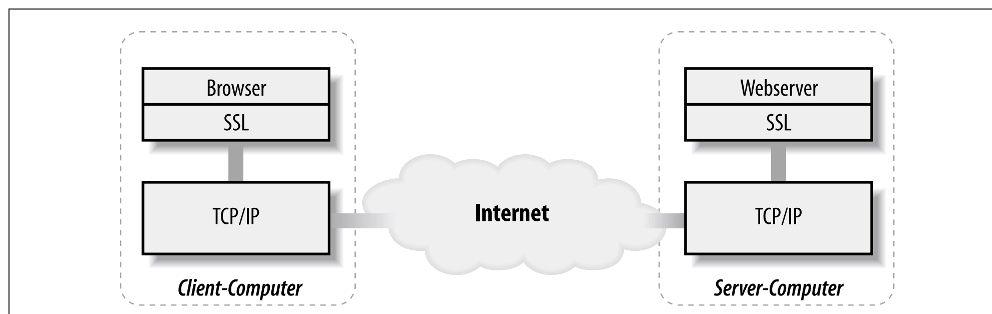

Abbildung 11-3: HTTP-Clients und -Server, SSL und die Netzwerkschicht, die TCP/IP implementiert

Weil SSL zwischen HTTP und TCP/IP gelagert ist, verschicken sichere Websites technisch gesehen eigentlich kein HTTP, zumindest nicht direkt über TCP. URLs für Ressourcen auf einem sicheren Server beginnen mit *https://*. Das steht für HTTP über SSL. Der Standard-Port für einen SSL-Dienst ist 443, nicht wie bei HTTP 80. Wenn ein Browser sich mit *https://secure.example.com* verbindet, erfolgt die TCP/IP-Verbindung über den Port 443 auf *secure.example.com*. Die meisten Browser und Webserver können SSL unterstützen, aber in die Konfiguration des Webservers müssen Schlüssel und Zertifikate eingeschlossen werden (und möglicherweise auch in die des Browsers, wenn eine Client-Zertifizierung erforderlich ist). Außerdem müssen Browser mit Zertifikaten von Stamm-Zertifizierungsstellen (CAs) vorkonfiguriert werden. CAs und Zertifikate behandeln wir später.

Cipher-Suites

Um einen Dienst anzubieten, dessen Ziele Vertraulichkeit, Unversehrtheit und Authentizität sind, nutzt SSL eine Verknüpfung kryptographischer Techniken. Dazu gehören Nachrichten-Prüfsummen, digitale Zertifikate und natürlich Verschlüsselung. Es gibt viele verschiedene Standardalgorithmen, die diese Funktionen implementieren, und SSL kann unterschiedliche Kombinationen einsetzen, um besonderen Anforderungen genügen zu können (z.B. wenn das Problem einfach nur darin besteht, dass die Verwendung einer Technik in bestimmten Staaten rechtlich nicht zulässig ist!).

Wenn eine SSL-Verbindung hergestellt wird, handeln Client und Server – basierend auf den gemeinsamen Fähigkeiten – die beste Kombination von Techniken aus, um sicherzustellen, dass ein höchstmögliches Maß an Sicherheit gegeben ist. Die Kombinationen der auszuhandelnden Techniken werden als *Cipher-Suites* bezeichnet.

SSL-Sessions

Wenn ein Browser sich mit einer sicheren Site verbindet, führt das SSL-Protokoll die folgenden vier Schritte aus:

1. Es wird eine Cipher-Suite ausgehandelt. Der Browser und der Server ermitteln die höchste unterstützte SSL-Version und die konfigurierten Fähigkeiten. Dann wird die stärkste Cipher-Suite ausgewählt, die von beiden Systemen unterstützt werden kann.
2. Zwischen dem Server und dem Browser wird ein geheimer Schlüssel ausgetauscht. In der Regel generiert der Browser einen geheimen Schlüssel, der mit Hilfe des öffentlichen Schlüssels des Servers mit einem Einweg-Verfahren (asymetrisch) verschlüsselt wurde. Das Geheimnis dieses Schlüssels kann nur der Server brechen, indem er es mit dem privaten Schlüssel entschlüsselt, der dem öffentlichen Schlüssel zugeordnet ist. Das geteilte Geheimnis wird dann als Schlüssel verwendet, um die versendeten HTTP-Nachrichten zu verschlüsseln und zu entschlüsseln. Diese Phase wird als *Schlüsselaustausch* bezeichnet.
3. Der Browser authentifiziert den Server, indem er das digitale X.509-Zertifikat des Servers analysiert. Häufig sind Browser von Zertifizierungsstellen im Vorhinein mit einer Liste von Zertifikaten versehen worden. Wenn das Zertifikat gültig ist, wird der Benutzer wird darauf aufmerksam gemacht, dass er auf eine sichere Site wechselt; kennt der Browser das Zertifikat nicht, wird der Benutzer gewarnt. Das geschieht üblicherweise, indem ein Dialog eingeblendet wird, mit dem der Benutzer gefragt wird, ob er weitermachen will, obwohl die Authentifizierung fehlgeschlagen ist.
4. Der Server untersucht das X.509-Zertifikat des Browsers, um den Client zu authentifizieren. Dieser Schritt ist optional und verlangt, dass alle Clients ebenfalls mit digitalen Zertifikaten ausgestattet wurden. Apache kann so konfiguriert werden, dass er das X.509-Zertifikat des Browsers so behandelt, als wäre es der Benutzername und das Passwort, die im `Authorization`-Feld eines HTTP-Headers kodiert sind. Im Web werden üblicherweise keine Client-Zertifikate verwendet.

Diese vier Schritte bieten eine kurze Zusammenfassung des Netzwerk-Verbindungsaufbaus zwischen Browser und Server, wenn SSL verwendet wird. Haben Browser und Server diese Schritte einmal abgeschlossen, können HTTP-Anfragen mit SSL verschlüsselt und an den Webserver geschickt werden.

Der SSL-Verbindungsaufbau läuft langsam, und würde er bei jeder HTTP-Anfrage durchgeführt, wäre die Leistung einer sicheren Website jämmerlich. Um die Leistung zu verbessern, nutzt SSL das Session-Konzept. Das ermöglicht es, eine einmal ausgehandelte Cipher-Suite, den geteilten geheimen Schlüssel und die Zertifikate für mehrere Anfragen zu verwenden. Eine SSL-Session wird von der SSL-Software verwaltet und ist nicht dasselbe wie eine PHP-Session.

Zertifikate und Zertifizierungsstellen

Ein signiertes *digitales Zertifikat* kodiert Informationen, damit die Sicherheit der Informationen und deren Signatur geprüft werden können. Die Informationen, die in einem Zertifikat enthalten sind, das für SSL verwendet wird, enthalten Daten zum Unternehmen und den öffentlichen Schlüssel des Unternehmens. Dem öffentlichen Schlüssel, der in einem Zertifikat enthalten ist, entspricht ein privater Schlüssel, der in den Webserver des

Unternehmens einkonfiguriert ist. Wenn Sie unseren Installationsanweisungen für Unix- oder Mac OS X-Plattformen in den Anhängen A bis C gefolgt sind, werden Sie sich erinnern, wie wir das Schlüsselpaar erzeugt und dem Webserver den privaten Schlüssel hinzugefügt haben.

Wenn eine SSL-Session aufgebaut wurde, verwendet der Browser den öffentlichen Schlüssel, um ein Geheimnis zu verschlüsseln. Das Geheimnis kann dann nur mit dem privaten Schlüssel entschlüsselt werden, der in den Server des Unternehmens einkonfiguriert ist. Verschlüsselungstechniken, die einen öffentlichen und einen privaten Schlüssel einsetzen, werden als *Einweg-Verschlüsselung* oder *asymmetrische Verschlüsselung* bezeichnet. SSL nutzt eine asymmetrische Verschlüsselung, um einen geheimen Schlüssel auszutauschen. Der geheime Schlüssel kann dann verwendet werden, um Nachrichten zu verschlüsseln, die über das Internet versandt werden.

Natürlich können Sie nicht darauf vertrauen, dass ein unbekannter Server tatsächlich das ist, was er zu sein vorgibt. Sie sind darauf angewiesen, dass Ihnen eine bekannte Autorität garantiert, dass der Server die Wahrheit sagt. Dieser Autorität müssen Sie dann vertrauen. Das ist die Rolle einer *Zertifizierungsstelle* (*Certification Authority*, CA). Jedes signierte Zertifikat enthält Daten zur CA. Die CA signiert ein Zertifikat digital, indem sie ihre eigenen Unternehmensdaten, eine verschlüsselte Prüfsumme des Zertifikats (die mit einer Technik wie MD5 erzeugt wurde) und den eigenen öffentlichen Schlüssel hinzufügt. Mit diesen kodierten Informationen kann sichergestellt werden, dass ein vollständig signiertes Zertifikat korrekt ist.

Es gibt Dutzende, wahrscheinlich sogar Hunderte von CAs. Man kann nicht erwarten, dass ein Browser (oder der Benutzer, der vom Browser mit einer Warnung konfrontiert wird) die digitalen Signaturen all dieser Autoritäten kennt. Der X.509-Zertifizierungsstandard löst dieses Problem, indem er es zulässt, dass die Signaturen ausgebender CAs selbst wieder digital von anderen CAs signiert werden, deren Signaturen wiederum von einer weiteren, noch vertrauenswürdigeren CA signiert sind. Irgendwann endet diese Kette der Signaturen bei der einer Stamm-Zertifizierungsstelle. Wie bereits erwähnt wurde, sind die Zertifikate der Stamm-CAs in der Regel bereits in der Browser-Software vorinstalliert. Außerdem ermöglichen es die meisten Browser dem Benutzer, seine eigenen vertrauenswürdigen Zertifikate hinzuzufügen.

Wenn Sie nicht für ein Zertifikat zahlen wollen oder eines zum Testen benötigen, können freie Zertifikate erzeugt und verwendet werden, um einen Webserver mit SSL zu konfigurieren. In den Anhängen A bis C zeigen wir für Unix- und Mac OS X-Plattformen, wie man freie selbst signierte Zertifikate erzeugt. Außerdem gibt es die Möglichkeit, von VeriSign unter *http://www.verisign.com/* freie Testzertifikate zu erhalten. Allerdings sind selbst signierte Zertifikate oder Testzertifikate normalerweise nur in eingeschränkten Umgebungen wie Unternehmensnetzwerken nützlich. Benutzer von sicherheitsrelevanten Applikationen im Internet werden ihnen nicht vertrauen. Deswegen werden Sie wahrscheinlich bezahlen müssen, um Ihr Zertifikat signieren zu lassen, bevor die Anwendung tatsächlich in der Produktion eingesetzt werden kann.