PHP/Anwendung und Praxis/Loginsystem

Aktualisierung und Erweiterung (Hashen der Passwörter) eines Artikels von Benjamin Wilfing (2005)

Ehemaliger Artikel im SelfHTML-Wiki.

Informationen zum Autor:

Name:
Jörg Reinholz
E-Mail:
joerg.reinholz@fastix.org
Homepage:
fastix.org

Wenn man Teile eines Internetauftritts vor neugierigen Augen schützen will, benötigt man ein wirksames System zur Zugangskontrolle. In vielen Fällen bietet sich hier die bekannte HTTP-Authentifizierung an, die man beispielsweise per `.htaccess`-Datei umsetzen kann. Diese Methode hat aber den Nachteil, dass es auch in modernen Browsern immer noch keine praktikable Möglichkeit gibt, sich wieder abzumelden. Dieser Artikel soll zeigen, wie man Dokumente auf relativ einfache Weise sinnvoll schützen kann. Voraussetzung dafür ist ein Webserver mit PHP-Unterstützung.

Inhaltsverzeichnis

Stand der Technik

Hinweis zu Strato

Der Massenhoster Strato verwendet eine recht eigentümliche Konfiguration, die dazu führt, dass das Skript an manchen Stellen an diese Besonderheiten angepasst werden musste. Da dieses mangels eines genau so konfigurierbaren Testsystems nicht überprüfbar ist, bitte ich um Nachricht, ob das so läuft (oder nicht). Kleiner Vorteil: Wenn es nicht läuft kann ich auch helfen, brauche aber eine vernünftige Fehlerbeschreibung.

Aktualität des Artikels

Da sich Verschlüsselungstechnologien (das Hashen zählt dazu) relativ schnell ändern: Dieser Artikel wurde im Februar 2015 verfasst und im August 2016 zuletzt geändert. Die Skripte sollten mit PHP 5.5 und neuer getestet wurde mit PHP 5.5.9, 7.0.2. Gezeigtes HTML ist Version 5.

Wozu das gut ist

Bis vor einigen Jahren hielt man den MD5-Algorhitmus für sicher. Das sind aber nur 16Byte (128Bit) pro Passwort. Nimmt jetzt ein Angreifer eine Liste mit, sagen wir, 1 Milliarde häufiger Passwörter, dann braucht er nur eine Datenbank mit 16 Milliarden Bytes für die Hashes und rund 8 Milliarden Bytes für die Passwörter. Das sind rund 24 Gigabyte (es wird womöglich etwas mehr sein). Das bedeutet aber, dass derartige Tabellen leicht zu speichern sind - denn die passen sogar auf einen billigen USB-Stick. Selbst ein Brute-Force-Angriff (bei dem alle möglichen Passwörter "durchgespielt" werden) ist mit aktueller Rechentechnik leicht möglich. Ein einfaches Hashen mit Verfahren wie MD5 oder sha geht inzwischen so schnell, da ist sogar die Notwendigkeit des Speicherns der "Rainbow-Tables" nicht mehr wirklich nötig, und ohne den Salt könnte man mit selbst auf Konsumer-Systemen speicherbaren Tabellen arbeiten, denn inzwischen haben deren Festplatten die Größe mehrerer Terabytes.

Wenn man in die Presse schaut findet man immer wieder Meldungen, dass Datenbanken mit Passwörtern in die Hände von Angreifern gelangt sind - und das Problem betrifft auch große Firmen von denen man meint, die sollten sich die nötige technische Intelligenz eigentlich kaufen können. Das findet aber oft aus Unwisssenheit, Profitgier (oder beidem) nicht statt und dann wundert man sich, wenn interessierte 13-jährige aus einer solchen Datenbank mit "gehashten" Passwörtern zumindest eine Vielzahl der von den Kunden genutzten Passwörter im Klartext ermitteln - und zwar auf einem vom Papa überlassenen, alten Rechner.

Umsetzung, Test und Download

Bitte beachten Sie: Das System wird hier des Umfangs wegen nur in den Grundzügen beschrieben. Testen können Sie das System hier. Es gibt dort auch einen Download mit einer Online-Verwaltung für Benutzer und Gruppen. Diese Software ist derzeit im BETA-Test.

Das System wird mittels Sessions (engl. Sitzung) umgesetzt. In PHP ist eine Session im Prinzip eine

Datei, die sich im Dateisystem des Webservers befindet. In ihr sind Daten gespeichert, auf die man über das spezielle Array `$_SESSION` zugreifen kann. Jeder Benutzer, der mit seinem Browser eine Seite aufruft, die Sessions benutzt, bekommt nun vom Server eine einmalige Session-ID zugewiesen, über die er identifiziert werden kann. Dadurch ist es möglich, Daten benutzerbezogen zu speichern, die während einer Sitzung wiederverwendet werden können. Anwendungsmöglichkeiten sind zum Beispiel ein Warenkorb beim Einkaufen im Internet oder eine Administrationsoberfläche für ein Content-Management-System (CMS).

Erst ein mal ein wenig "praktische Theorie": Passwörter hashen

Die Password-Funktionen seit PHP 5.5.0

Generell ist die neuere Funktion password_hash() anderen wie crypt vorzuziehen. (Wobei crypt sehr wohl schon lange das Hashen mit der "blowfish"-Methode kann.) password_hash() kommt mit den Funktionen password_verify() und password_needs_rehash(). Die Funktion ist grob dargestellt die folgende:

So hashen Sie ein Passwort:

```
$hash=password_hash($password, PASSWORD_DEFAULT);
```

So prüfen Sie das Passwort gegen den Hash:

```
if ( password_verify( $password, $hash ) ) { ... }
```

Ein stilles Update können Sie auch durchführen:

Nach einem erfolgreichen Login führen Sie folgendes aus:

```
if ( password_needs_rehash($hash, PASSWORD_DEFAULT) ) {
    $newHash = password_hash( $password, PASSWORD_DEFAULT );
    /* Jetzt den neuen Hash speichern! */
};
```

Die Funktion password_hash wird einen hash (das ist eine Art "Einweg-Verschlüsselung") mit dem Blowfish-Algorithmus erzeugen. Dazu ermittelt diese zunächst einen zufälligen, sogenannten "salt", der immerhin 22 Zeichen lang ist und aus allen Ascii-Buchstaben, den Ziffern und Satzzeichen bestehen darf. Es gibt bei dieser Länge (26+26+10+5)^22 = 14.915.769.363.385.151.583.217.201.855.136.979.828.889 (~1.5 * 10^40) also eine so gewaltige Zahl verschiedener "Salts", dass sich diese nicht speichern lassen. Die dazu gehörende Anzahl an Rainbow-Tables erst recht nicht. Dann wird Folgendes gemacht: Das "salt" wird dem Password vorangestellt und diese Kombination wird mit dem Blowfisch-Algorithmus gehasht. Das "Salzen" und "Hashen" findet dann in der unter "Kosten" angegeben Zahl von Runden statt: steht da 05, dann werden 2^4 (das Minimum), also 16 - steht da aber 31 (das Maximum) dann werden 2.147.483.648 Runden gedreht - was sehr sicher klingt aber leider auch zu lange dauern kann. Der Hash wird wie folgt zurück gegeben: $_METHODE_$_ITERATIONEN_$_SALT_$_EIGENTLICHER_HASH.

Das Login-System

Das hier beschriebene Loginsystem umfasst die folgenden 8 Dateien

- das Skript `login.php`,
- das Skript `logout.php`,
- das Skript `auth.php`,
- die Apache-spezifische Steuerungs-Datei `.htaccess`,
- das Skript `sendFile.php` - für das Ausliefern von Dateien, welche keine PHP-Skripte sind
- die Ressource `.htpasswd` - mit den gehashten Passwörtern
- die Ressource `.htgroup` - mit den Gruppen und Benutzern
- das Skript bzw. Ressource `auth_config.php` - das die ist einzige Datei, welche für die Konfiguration bearbeitet werden muss. (Diese enthält auch schon Voreinstellungen für die Software zur Manipulation der Ressourcen.)

... welche sich hier der Einfachheit halber alle im selben Verzeichnis befinden. (Alle Dateien, die mit '.ht' beginnen sollten sich eigentlich tunlichst nicht in einem Verzeichnis befinden, dessen Inhalte der Webserver ausliefert - in der Datei `auth_config.php` ist das dann anders zu konfiguren. Die Vergabe eines mit ".ht" beginnenden Namens ist nur ein Notbehelf: Ein Apache-Webserver wird diese nicht ausliefern, wenn es nicht anders konfiguriert wird. Die Datei `login.php` beinhaltet sowohl das Formular zum Anmelden als auch die Routinen, um die Benutzerdaten zu verarbeiten und bei erfolgreicher Anmeldung entsprechende Daten in die Session zu speichern. Die Datei `logout.php` meldet den Benutzer ab, indem sie die Sitzungsdaten zerstört und `auth.php` enthält die Überprüfung, ob der Benutzer aktuell angemeldet und somit berechtigt ist das angeforderte Dokument anzusehen. Diese Datei `auth.php` wird mit `require_once` in jedes zu schützende PHP-Dokument eingebunden, damit bei einem Fehler bei der Einbindung (z.B. durch einen Tippfehler) nicht etwa nur eine Warnung angezeigt wird.

Die Ressource `login.php`

Beispiel: login.php

```
<?php
  ini_set('php_value output_buffering', '1');
  ini_set('session.use_trans_sid', '0');
  ini_set('session.use_cookies' , '1' );
  ini_set('session.use_only_cookies' , '1');

  #Lesen der Konfiguration
  require_once('auth_config.php');


  if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    show_login('', false);
    exit;
  }

  // Das Feld mit dem Passwort hat einen zufälligen Name
```

```
// damit das Passort nicht gespeichert werden kann.
if ( isset($_POST['password_field_name']) ) {
    $_POST['passwort'] = trim($_POST[$_POST['password_field_name']]);
}

// Simple Eingabeprüfung auf Übertragung des Benutzernamens und des
// Passwortes:
if ( ! isset ($_POST['username']) )   {
    show_login('', 'Benutzername nicht übertragen.');
    exit;
}
if ( '' == trim($_POST['username']) ) {
    show_login('', 'Benutzername leer.');
    exit;
}
if ( ! isset ($_POST['passwort']) )   {
    show_login('', 'Passwort nicht übertragen.');
    exit;
}
if ( '' == trim($_POST['passwort']) ) {
    show_login('', 'Passwort leer.');
    exit;
}

// Benutzername und Passwort werden überprüft

// Schritt 1:
// Im Beispiel seien die Zugangsdaten in einer Textdatei gespeichert, welche
// im wesentlichen dem Aufbau einer Datei entspricht, wie diese der Apache mit
// der Methode htpasswd verwendet.
// Das wäre hier mit den Benutzernamen foo und bar und dem identischen(!)
// Passwort
// 'GeHeim' etwas wie:

// foo:$2y$05$AoeUm/2hL5sdySuNuH0CmeSCZN7DneDdmU3thyrFWgiboh3FYQ0ae
// bar:$2y$05$9971LHpPxCmZikyi3WqEd.1h8GE6QKfqT4RwS/FDINRqyzLcEHy.C

// Weiter wird der Benutzername beim Speichern stets klein geschrieben und
// man muss absichern,dass keine Leerzeichen am Beginn oder Ende des Strings
//mitgeliefert wurden:

$_POST['username']=strtolower(trim($_POST['username']));

// Wir kapseln aber das Abholen des Passwortes, damit auch Datenbanken
// verwendet werden können.
// Dazu muss unten die Funktion  getHashedPassword($username) angepasst
// werden.

$HashedPassword = getHashedPassword($_POST['username']);
if (false == $HashedPassword ) {
    show_login('', 'Den Benutzer gibt es nicht.');
    exit;
}
// Schritt 2:  Überprüfen des Passwortes
 if ( ! password_verify (trim($_POST['passwort']), $HashedPassword) ) {
     show_login(trim($_POST['username']), 'Passwort falsch');
     exit;
 }

 // else:
 // Registrierung der Daten in der Session und
```

```
   // Weiterleitung zur geschützten Startseite
   session_start();
   session_unset();
   session_regenerate_id(true);

   if ( isset($_POST['register_ip']) &&  $_POST['register_ip'])  {
      $_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
   }
   $_SESSION['username']      = $_POST['username'];
   $_SESSION['groups']        = getGroups($_POST['username']);
   $_SESSION['last_action']   = date('U');
   $target = $_SERVER['REQUEST_SCHEME'] . '://' . $_SERVER['HTTP_HOST'] .
rtrim(dirname($_SERVER['SCRIPT_NAME']), '/') . '/index.php';
   header('Location: ' . $target , true, $_SERVER['SERVER_PROTOCOL'] ==
'HTTP/1.1' ? 303 : 302);
   exit;

  // benötigte Funktionen

  function getHashedPassword($username) {
       $arRows = file(PASSWORD_FILE);
       unset($csv); // Speicher sparen
       foreach($arRows as $row) {
            $row=trim($row);
            //Ausschließen von Kommentaren und Leerzeilen sowie Zeilen
            // ohne Trenner:
            if ( $row && '#' != $row{0}  && strpos($row, ':') ) {
                // Aufspalten von Benutzername und Passwort
                $arPair = explode(HTPASSWD_SEPARATOR, $row, 3);
                if ( trim($arPair[0]) == $username  ) {
                   return trim($arPair[1]);
                }
            }
       }
       // wenn die Funktion hier noch aktiv ist, dann gibt es den Benutzer
       // nicht;
       return false;
  }

  function getGroups($username) {
    // gibt einen Hash (Hier gemeint: Array mit benamten Elementen)
    // mit allen Gruppen, zu denen der sich anmeldende Benutzer gehört, zurück
    $r=array();
    $r['standard-user']=true;
    $arRows = file(GROUP_FILE);
    foreach($arRows as $row) {
       $row=trim($row);
       if ($row != '' && '#' != $row{0} && strpos($row, ':') ) {
          list($group, $rest)=explode(HTGROUP_SEPARATOR, $row);
          $arUsers=explode(HTGROUP_USER_SEPARATOR, $rest);
          foreach ($arUsers as $s) {
            $s=trim($s);
            if ($s == $username) {
              $r[$group]=true;
            }
          }
       }
    }
    return $r;
  }
```

```
  function show_login($username='', $explain=false) {
      if ($explain) {
         $explain = '<p> (Grund: '. $explain . ')</p>';
      }
      $username = htmlspecialchars(str_replace('"', '&quot;', $username));

      // Zufälliger Name für das Passwort-Feld:
      $password_field_name = '';
      $chars = 'abcdefghijklmnopuvwxyzABCDEFGHIJKLMNOPUVWXYZ';
      $rand_max= strlen($chars) - 1;
      for ($i=0; $i<10; $i++) {
         $password_field_name .= $chars{rand(0, $rand_max)} ;
      }
      $hostname=$_SERVER['HTTP_HOST'];
      header('Expires: Thu, 01-Jan-70 00:00:01 GMT');
      header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
      header('Cache-Control: no-store, no-cache, must-revalidate');
      header('Cache-Control: post-check=0, pre-check=0', false);
      header('Pragma: no-cache');
      echo <<<EOF
<!doctype html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta charset="utf-8">
    <meta name="robots" content="noindex"><!-- soll in Suchmaschinen nicht
sichtbar sein -->
    <title>$hostname: Geschützter Bereich</title>
  </head>
  <body>
    <h1>Bitte anmelden:</h1>
    $explain
    <form action="login.php" method="POST">
      <fieldset>
        <legend>Login</legend>
        <label for="username">Username: <input type="text" name="username"
id="username" value="$username" required autofocus></label><br>
        <label for="$password_field_name">Passwort: <input type="password"
name="$password_field_name" id="$password_field_name" required></label><br>
        <label for="register_ip"><input type="checkbox" checked="checked"
name="register_ip" id="register_ip" value="1"> Anmeldung an IP-Adresse
binden</label><br>
        <input type="submit" value="Anmelden">
      </fieldset>
      <input type="hidden" name="password_field_name"
value="$password_field_name">
    </form>
  </body>
</html>
EOF
;
}
```

Der PHP-Programmcode ist für den Anmeldeprozess zuständig. Das in die index.php (und jedes weitere PHP-Skript) includierte `auth.php` prüft zuerst, ob der Benutzer schon angemeldet ist. Ist das nicht der Fall, wird ein Formular angezeigt, welches Benutzername und Passwort über die Methode `POST` wieder an die Datei `login.php` sendet, um dann wiederum weiterverarbeitet zu werden. Hier sind sowohl Benutzername als auch Passwort in einer Datei `.htpasswd` gespeichert

- denkbar wäre aber zum Beispiel auch die Abfrage von Benutzerdaten aus einer Datenbank. Deshalb wurde auch das Holen des gehashten Passwortes in einer Funktion untergebracht. Diese ist hierdurch leicht austauschbar.

Nach dem Abholen des geshashten Passwortes wird das eingegebene Passwort ebenfalls mit der selben Methode und dem selben Salt gehasht und beide Hashs werden verglichen, denn dass eigentliche Passwort kann nicht wieder hergestellt werden, weshalb viele im Zusammenhang von einer "Einwegverschlüsselung" sprechen. Stimmen die Hashes überein, dann wurden Benutzername und Passwort korrekt eingegeben. Der Benutzername wird in der am Anfang des Skripts geöffneten Sessionvariable gespeichert und anhand dieser wieder erkannt. Dieses Konzept ist z.B. auf Gruppen und also nutzerabhängige Berechtigungen erweiterbar.

Aus der Datei `.htgroup` werden die Gruppen und deren Benutzer ausgelesen. Findet sich in letzteren eine Übereinstimmung, so wird die Gruppe ebenfalls in der Session gespeichert.

Im nächsten Schritt wird der Benutzer auf die Datei `index.php` weitergeleitet. Dies geschieht mittels der Funktion `header('Location: ...')`. Da die Location-Anweisung einen vollständigen URI erwartet, man diesen vielleicht aber aus Bequemlichkeit nicht für jedes Projekt ändern will oder den künftigen URI noch nicht kennt, wird er aus Servervariablen zusammengebaut, so dass man nur noch den letzten Teil an seine Bedürfnisse anpassen muss. Im Beispiel befindet sich die Datei `index.php` im selben Verzeichnis wie `login.php`.

Waren Benutzername oder Passwort aber falsch, wird wieder das Login-Formular angezeigt - der Benutzer muss aber, falls sein Benutzername korrekt war, nur das Passwort erneut eingeben.

Der HTTP Statuscode `303 See Other` wird hier aus Gründen der Standardkonformität gesendet: Bei einem Location-Header sendet PHP standardmäßig den Statuscode 302, nur ist hier nicht definiert, ob der nachfolgende Weiterleitungsrequest per POST oder GET getätigt werden soll. HTTP/1.1 wurde daher unter anderem um die Statuscodes 303 und 307 erweitert, die genau das festlegen. Ältere Browser, die nur HTTP/1.0 implementiert haben, bleiben bei der Abfrage nach `$_SERVER['SERVER_PROTOCOL']` außen vor, bekommen so den obligatorischen Statuscode 302 und reagieren darauf, wie sie programmiert wurden. In fast allen dieser älteren Browser wird die nächste Seite dann per GET angefordert.

Das Login-Formular wird mit dem HTML-Code `<meta name="robots" content="noindex">` versehen, damit Suchmaschinen das Login-Formular gar nicht erst aufnehmen. Mögliche Angreifer werden also nicht durch Suchmaschinen mit der Information versorgt, dass sich da ein Anmeldeformular befindet. Dieses Vorgehen wird Angriffsversuche nicht unterbinden - ist aber, weil viele Angreifer tatsächlich Suchmaschinen benutzen um Angriffsziele auszumachen, durchaus geeignet um deren Zahl zu reduzieren.

Passwort-Datei `.htpasswd`

Eine Datei mit Benutzernamen und Passwörtern für die hier vorgestellten Skripte sähe (ungefähr) wie folgt aus:

Beispiel: Ressource .htpasswd

```
#Benutzer:Hash(Passwort)
root:$2y$05$koUjHnB4PSch7eCcJtvm0uPbjJPbkO0NnAypgFlqF2TqO/4FLvXL2
foo:$2y$05$zvl33PexEI9hqPJzw8VIn.CsyLdM5U2.YpITGzB/VF8N.U5VeCIqi
```

und würde, weil diese den gleichen Aufbau hat, auch die native Möglichkeit des Apache (ab 2.4!) für die Authentifizierung mit htaccess unterstützen. Wer zudem auf einen Rechner mit einer halbwegs vollständigen Apache 2.4-Installation (mit den Apache-Tools) hat, der kann die Datei auch wie folgt erzeugen:

```
user@host:~$ htpasswd -nB foo
New password: (EINGABE: GeHeim)
Re-type new password: (EINGABE: GeHeim)
foo:$2y$05$zvl33PexEI9hqPJzw8VIn.CsyLdM5U2.YpITGzB/VF8N.U5VeCIqi
user@host:~$
```

und ggf. die Ausgabe auch gleich umleiten:

```
user@host:~$ htpasswd -nB foo >> .htpasswd
New password: (EINGABE: GeHeim)
Re-type new password: (EINGABE: GeHeim)
user@host:~$
```

Benutzername und gehashtes Passwort stehen dann in der Datei. Weitere werden jeweils angehängt.

Gruppen-Datei `.htgroup`

Eine Datei mit Gruppen und zugehörigen Benutzernamenfür die hier vorgstellten Skripte sähe (ungefähr) wie folgt aus:

Beispiel: Ressource .htgroup

```
#Gruppe:Benutzer,Benutzer ...
adm:foo,root
root:bar,root
```

Es ist also gut zu erkennen, dass der erste Abschnitt den Gruppenname enthält, dann folgt ein Doppelpunkt und die Liste der Mitglieder. Die hat das Komma als Trenner.

Die Ressource `logout.php`

Beispiel: Skript logout.php

```
<?php
  session_start();
  session_destroy();
  # Nur in harten Fällen benutzen, wenn der Server session_destroy() nicht
korrekt unterstützt:
  # unlink ( SESSION_FILE_DIR . '/sess_' . session_id());
```

```
  header('Location: http://' . $_SERVER['HTTP_HOST'] .
rtrim(dirname($_SERVER['SCRIPT_NAME']), '/') . '/login.php');
```

Die Datei `logout.php` ist einzig und allein dafür zuständig, die Session - und damit die Information, dass der Benutzer angemeldet ist - zu zerstören. Danach wird der Benutzer wieder zum Anmeldeformular weitergeleitet.

Die Ressource `auth.php`

Beispiel: Skript auth.php

```
<?php
  require_once('auth_config.php');
  session_start();

  if (
           ( ! ( isset($_SESSION['username']) && $_SESSION['username']) )
        or ( isset($_SESSION['ip']) &&  ! $_SESSION['ip'] ==
$_SERVER['REMOTE_ADDR'] )
        or ( ($_SESSION['last_action'] + SESSION_MAX_IDLE_TIME) < time() )
  ) {
     header('Status: 403 Forbidden');
     header('Location: ' . $_SERVER['REQUEST_SCHEME'] . '://' .
$_SERVER['HTTP_HOST'] . rtrim(dirname($_SERVER['SCRIPT_NAME']), '/') .
'/login.php');
     exit;
  }

  if ( defined('NEED_GROUPS') && '' != trim(NEED_GROUPS) ) {
     if (! check_user_has_group(NEED_GROUPS) ) {
        show_forbidden ();
        exit;
  }  }

  if ( defined('DENIED_GROUPS') && '' != trim(DENIED_GROUPS) ) {
     if (check_user_has_group(DENIED_GROUPS) ) {
        show_forbidden ();
        exit;
  }  }

  if ( check_user_forbidden () ) {
        session_destroy();
        show_forbidden ();
        exit;
  }

  $_SESSION['last_action'] = time();
  // Kein Exit, da das aufrufende Skript weiter arbeiten muss!


  // Funktionen:

  function check_user_has_group($list) {
     $ar=explode(',', $list);
     foreach ($ar as $group) {
        $group=trim($group);
        if ( isset($_SESSION['groups'][$group]) && $_SESSION['groups'][$group])
        {
```

```
            return true;
      }  }
      return false;
   }

   function check_user_forbidden() {
      $ar=file(DENIED_USERS_FILE);
      foreach ($ar as $line) {
         if ( $_SESSION['username'] == trim($line) ) {
            return true;
      }  }
      return false;
   }

function show_not_found () {
      header('Status: 404 Not Found');
      header('Expires: Thu, 01-Jan-70 00:00:01 GMT');
      header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
      header('Cache-Control: no-store, no-cache, must-revalidate');
      header('Cache-Control: post-check=0, pre-check=0', false);
      header('Pragma: no-cache');
      print <<<EOF
<html>
  <head>
     <title>404 Not found</title>
     <meta name="robots" content="noindex">
     <meta charset="utf-8">
  </head>
  <body>
    <h1>404 Not found</h1>
    <p>Die Datei wurde nicht gefunden.</p>
  </body>
</html>
EOF
;
exit;
}

function show_forbidden () {
      header('Status: 403 Forbidden');
      header('Expires: Thu, 01-Jan-70 00:00:01 GMT');
      header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
      header('Cache-Control: no-store, no-cache, must-revalidate');
      header('Cache-Control: post-check=0, pre-check=0', false);
      header('Pragma: no-cache');
      print <<<EOF
<html>
  <head>
     <title>403 Forbidden</title>
     <meta name="robots" content="noindex">
     <meta charset="utf-8">
  </head>
  <body>
    <h1>403 Forbidden</h1>
    <p>Sie haben kein Recht, dies zu tun.</p>
  </body>
</html>
EOF
;
exit;
}
```

Dies ist die Ressource, die später in jede zu schützende Seite eingebunden werden muss. Es wird dann immer überprüft, ob die Sessionvariable `angemeldet` existiert und ihr Wert `true` ist. Falls das nicht der Fall ist, wird der Benutzer wieder zum Anmeldeformular weitergeleitet. Wichtig ist hier die `exit`-Anweisung nach der Weiterleitung, damit kein weiterer Code ausgeführt werden kann und das Skript sofort beendet wird.

Weiterhin wird, falls vor dem Aufruf eine Konstante NEED_GROUPS und/oder DENIED_GROUPS mit jweils einer Liste von Gruppen definiert wurde geprüft, ob der Benutzer zugelassen wird. Ist NEED_GROUPS gesetzt, so erhalten die Mitglieder dieser Gruppe Zugriff, es sei denn der User ist auch Mitglieder einer der in DENIED_GROUPS gelisteten Gruppen. Die Ressource roots_und_admin.php zeigt, wie das Definieren geht.

Zuletzt wird noch geprüft, ob der identifizierte Benutzer in der als einfache Liste ausgeführten Sperrdatei vorkommt. Trifft dieser oder einer der vorherigen Auschlussgründe nicht zu, wird schlussendlich das aufrufende Skript weiter ausgeführt.

Die geschützte Datei `index.php`

Beispiel: Skript index.php

```
<?php require_once('auth.php'); ?>
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Geschützter Bereich</title>
  </head>
  <body>
    <h1>Herzlichen Glückwunsch, <?php echo
htmlspecialchars($_SESSION['username']); ?> </h1>
    <p>Sie sind nun angemeldet.</p>
    <p>Sie können sich auch wieder <a href="logout.php">abmelden</a>.</p>
  </body>
</html>
```

Die erste und einzige Anweisung, die eine geschützte Seite enthalten muss, ist der Befehl zum Einbinden der Datei `auth.php`, nämlich mit `require` in der ersten Zeile. Wenn Sie mehrere Dokumente schützen wollen, müssen Sie diese Zeile einfach am Anfang jeder Datei einfügen. Solange der Benutzer sich nicht explizit abgemeldet hat - ein Link zum Abmelden sollte natürlich immer vorhanden sein - ist keine weitere Eingabe des Passworts nötig.

Doch das geht noch besser. Man kann nämlich auch eine Gruppe definieren, welche den Zugriff haben soll:

Die geschützte Datei `roots_und_admin.php`

Beispiel: Skript roots_und_admin.php

```
<?php define('NEED_GROUPS','root,adm'); require_once('auth.php'); ?>
<!doctype html>
...
```

Die notwendigen Gruppenmitgliedschaften werden also in dem Ausdruck `define('NEED_GROUPS','root,adm')` einfach notiert. Hier wären dies die Gruppen "root" und "adm". Wer in diesen Gruppen Mitglied ist, der kann das Skript erfolgreich aufrufen. Für die Feststellung sorgt übrigens die Funktion `check_user_has_group()` in der `auth.php`. Diese lässt sich gut nutzen um z.B. einen Link anzubieten oder nicht.

Die Ressource `.htdeniedusers`

Eine weitere Datei ist diejenige, in welcher die gesperrten Benutzer einfach als Liste notiert werden: Ein Benutzer, eine Zeile:

Beispiel: Ressource .htdeniedusers

```
#denied users per row
denni
```

In der auth.php wird geprüft, ob der Benutzer drin steht. Das macht die Funktion `check_user_forbidden()`. Dieses Vorgehen hat den Vorteil, dass der Besucher bei nächsten Aufruf einer geschützten Ressorce auch außen vor bleibt, dass also nicht erst das nächste Login fehlschlägt.

Die Datei bzw. Ressource `auth_config.php`

In den Skripten `auth.php` und `login.php` und wird zunächst die Ressource `auth_config.php` geladen. Dies enthält insbesondere **einige anpassbare Voreinstellungen**. Das gezeigte Beispiel enhält aber auch schon Voreinstellungen, wie z.B. hinsichtlich der nicht löschbaren Benutzer für die Software, mit welcher die Benutzeradministration erfolgen kann.

Beispiel: Skript auth_config.php

```
<?php
// Konfiguration
  // Error-Reporting:
  // Einstellung für Setup/Entwicklung:
  error_reporting(E_ALL);
    // Einstellung für Produktionsumgebung/Betrieb:
  #error_reporting(NULL);

  // Maximale Spanne der UNTÄTIGKEIT in Sekunden
  // 60=1min   300=5min 900=15min 1800=30min 3600=1h
  define ( 'SESSION_MAX_IDLE_TIME', 900 );

  // Das Verzeichnis für die Session-Dateien. Belässt man das originale kann es
  // sein, dass PHP die Sessions zu früh "wegräumt":
  // Wenn das keine Software für Sie macht, dann müssen das Verzeichnis anlegen
```

```
  // und dafür sorgen, dass es beschreibbar ist.
  define ( 'SESSION_FILE_DIR', __DIR__ . '/sessions' );

  // Wenn Sie einen eigenen Server haben (kein Shared Hosting!) KANN es
  // sinnvoll sein diesen Wert auf false zu setzen, dann kann jeder Benutzer mit
  // den Rechten der Gruppe des Webservers (oft: www-data) die Session-Dateien
  // ansehen und z.B. löschen.
  // Im Zweifelsfall auf true setzen.
  define('SHARED_HOSTING', true);

  // Speichermethode, hier Text/CSV
  /* gilt für
  * Passwort-Datei
  * Gruppen-Datei
  * LastLogin-Datei
  Gültige Einträge: csv
  Geplante Einträge: json|database (Für künftige Versionen)
  */
  define ( 'STORAGE_METHOD', 'csv' );
  define ( 'STORAGE_DIR', __DIR__ . '/' );

  // Passwort-Datei
  define ( 'PASSWORD_FILE', STORAGE_DIR . '.htpasswd' );

  // Gruppen-Datei
  define ( 'GROUP_FILE', STORAGE_DIR . '.htgroup' );

  // Denied-Datei:
  define ( 'DENIED_USERS_FILE', STORAGE_DIR . '.htdeniedusers' );

  // Benutzer, die nicht gelöscht werden können. Liste, mit Komma getrennt
  define ( 'USERS_NODELETE', 'root,adm' );
  // Gruppen, die nicht gelöscht werden können. Liste, mit  Komma getrennt
  define ( 'GROUPS_NODELETE', 'root,adm' );

  define ( 'NAMES_PATTERN', '[0-9A-Za-z@_.-]{3,}' );
  define ( 'NAMES_PATTERN_DESCR','Gültig sind alle Buchstaben aus dem ASCII-
Zeichensatz und Ziffern sowie die Zeichen @, -, _  und der Punkt, die Eingabe
muss mindestens 3 Zeichen lang sein:' );

  define('MIN_PASSWORD_LENGTH', 8);
  $password_patterns[]='.{'. MIN_PASSWORD_LENGTH .'}'; # Länge: Mindestens
MIN_PASSWORD_LENGTH Zeichen
  $password_patterns[]='[A-Z]{1,}'; # gr. Buchstaben
  $password_patterns[]='[a-z]{1,}'; # kl, Buchstaben
  $password_patterns[]='[0-9]{1,}'; # Ziffern
  $password_patterns[]='[^A-Za-z0-9]{1,}'; # Sonderzeichen

  define ( 'PASSWORD_PATTERN_DESCR','Das Passwort muss mindestens ' .
MIN_PASSWORD_LENGTH . ' Zeichen lang sein und Buchstaben, Ziffern sowie
Sonderzeichen enthalten, es darf nicht mit einem Leerzeichen beginnen oder
enden.' );

  // Trennzeichen der Passwortdatei, default ist ":"
  define( 'HTPASSWD_SEPARATOR' , ':');
  // Trennzeichen der Extra-Informationen in der Passwortdatei, default ist ","
  define( 'HTPASSWD_EXTRA_SEPARATOR' , ':');

  // Trennzeichen der Gruppendatei zwischen Gruppe und Benutzerliste, default
ist ":"
  define( 'HTGROUP_SEPARATOR' , ':');
```

```
  // Trennzeichen der Gruppendatei zwischen den Benutzern, default ist ","
  define( 'HTGROUP_USER_SEPARATOR' , ',');



##############################################################################
#
# AB HIER NICHTS MEHR ÄNDERN!
# Prüfungen, automatische
# Einstellungen
#
##############################################################################


  # Kompatibilität zu älteren Apache/PHP-Versionen, welche
$_SERVER['REQUEST_SCHEME'] nicht liefern:
  if (! isset($_SERVER['REQUEST_SCHEME']) ) {
     if ( isset($_SERVER["HTTPS"]) && $_SERVER["HTTPS"] ) {
       $_SERVER['REQUEST_SCHEME']='https';
     } else {
       $_SERVER['REQUEST_SCHEME']='http';
     }
  }

  # zwingende Rechteeinschränkung für shared-Hosting
  if (SHARED_HOSTING) {
    define('AUTH_UMASK',    0077);
    define('AUTH_DIR_MOD',  0700);
    define('AUTH_FILE_MOD', 0600); # Auf Vorrat für die künftige Benutzung :)
  } else {
    define('AUTH_UMASK',    0007);
    define('AUTH_DIR_MOD',  0770);
    define('AUTH_FILE_MOD', 0660); # Auf Vorrat für die künftige Benutzung :)
  }

  define ('JSON_PASSWORD_PATTERN', json_encode($password_patterns) );

  if (STORAGE_METHOD == 'csv' or STORAGE_METHOD == 'json') {
    if ( '' == STORAGE_DIR ) {
      trigger_error('UPS! STORAGE_DIR darf nicht leer sein. Setzen Sie einen
Punkt, wenn Sie die Daten WIRKLICH im aktuellen Verzeichnis speichern wollen!',
E_USER_ERROR);
    }
    $ar=array(PASSWORD_FILE, GROUP_FILE, DENIED_USERS_FILE);
    foreach ($ar as $file) {
      if (! is_file(PASSWORD_FILE) ) {
        trigger_error('UPS! '.$file.' gibt es nicht.', E_USER_ERROR);
      }
      if (! is_readable(PASSWORD_FILE) ) {
        trigger_error('UPS! '.$file.' kann nicht gelesen werden.',
E_USER_ERROR);
      }
    }
  }


if ( defined('SESSION_FILE_DIR') ) {
    if (! is_dir( SESSION_FILE_DIR ) ) {
       $dummy=umask(0077);
       if (! mkdir(SESSION_FILE_DIR . '/', 0700, true) ) {
         trigger_error('Fatal: Unmöglich, das Verzeichnis für die Session-
```

```
Dateien anzulegen.', E_USER_ERROR);
       }
       if (! chmod( SESSION_FILE_DIR, 0700 ) ) {
          trigger_error('Fatal: Unmöglich, die Rechte für das Verzeichnis mit
Session-Dateien zu setzen.', E_USER_ERROR);
       }
       if (! file_put_contents(SESSION_FILE_DIR . '/' . '.htaccess', 'deny from
all') ) {
          trigger_error('Fatal: Unmöglich, das Verzeichnis für die Session-
Dateien zu sperren.', E_USER_ERROR);
       }
    }
    if (! is_writable(SESSION_FILE_DIR) ) {
      trigger_error('Fatal: Es ist unmöglich die Session-Dateien anzulegen.',
E_USER_ERROR);
    }
}

ini_set('session.gc_maxlifetime', SESSION_MAX_IDLE_TIME);
ini_set('session.save_handler', 'files');
session_save_path(SESSION_FILE_DIR);
ini_set('output_buffering', '1');
ini_set('session.use_trans_sid', '0');
ini_set('session.use_cookies' , '1' );
ini_set('session.use_only_cookies' , '1');
ini_set('session_cache_limiter', 'private');
header('access-control-allow-origin: ' . $_SERVER['REQUEST_SCHEME'] . '://' .
$_SERVER['SERVER_NAME'] . '/');
session_set_cookie_params (SESSION_MAX_IDLE_TIME);
/* ?> Nicht setzen, sonst wird ggf. ein oder mehrere Zeichen gesendet, was
Weiterleitungen stört */
```

Schutz von Ressourcen, welche keine PHP-Skripte sind:

Die hier beschriebene Methode bringt (bisher) gewisse Einschränkungen und Nachteile mit sich. Beispielsweise können im Gegensatz zur HTTP-Authentifizierung komplette Verzeichnisse nicht ohne weiteres geschützt werden, der Schutz beschränkt sich lediglich auf Dateien, die von PHP geparst werden. Aber genau das lässt sich ändern. Voraussetzung für das hier gezeigte ist ein Apache-Webserver mit dem Modul mod_rewrite und Sie müssen ggf. die Erlaubnis haben, eine Datei .htaccess anzulegen, die der Server dann auch beachtet. Hier ist ein Beispiel:

Beispiel: Resource .htaccess

```
# Es erfolgt mit voller Absicht keine Prüfung, ob mod_rewrite geladen ist!
# Denn es ist besser, einen 500er Fehler zu haben als die Dateien
# unbemerkt ungeschützt auszuliefern:
RewriteEngine On

#Versteckte Dateien werden verboten:
RewriteRule "^.*/\."           - [F]
RewriteRule "^\."              - [F]

#Niemals Ändern:
RewriteRule "^.*/\.ht"          - [F]
RewriteRule "^\.ht"             - [F]

#Directory Index (, ggf. mit Parametern)
RewriteRule "^$"               ./index.php [L]
```

```
RewriteRule "^[?&].*$"          ./index.php [L]

#PHP-Dateien kontrollieren sich selbst
RewriteRule "^(.*\.php)$"       - [L]
RewriteRule "^(.*\.php[?&].*)$" - [L]

#Ressourcen, die nicht durch sendFile.php kontrolliert werden
RewriteRule "^interior/(.*)$"   - [L]

#Alles andere liefert die Datei sendFile.php aus
RewriteRule "^(.*)$" ./sendFile.php?file=$1 [L]
```

Die Datei sendFile.php stellen wir auch kurz vor:

Beispiel: Skript sendFile.php

```
<?php
require_once('auth.php');
if ( ! empty($_GET['file']) ) {

    //// Dateiüberprüfung:

    // nur Verzeichnisse unterhalb von ./ sollen zulässig sein
    // Auflösen des Pfades zu einem absoluten Pfad
    // der übergebene Dateiname ist immer vergiftet ('tainted')
    $getFileTainted = realpath('./' . $_GET['file']);

    //// Abbruchbedingung:
    // $_GET['file'] ist  wenn das Ziel nicht existiert FALSE:
    if ( ! $getFileTainted ) {
        show_not_found ();
        exit;
    }

    //// Abbruchbedingung:
    // Ausschließen aller Dateien und Verzeichnisse die mit .ht beginnen
    if ( false !== strpos($getFileTainted, '/.ht') ) {
        show_forbidden ();
        exit;
    }

    //// Abbruchbedingung:
    // Für den Vergleich wird der absolute Pfad des aktiellem Verzeichnisses
benötigt:
    $thisDir = realpath('./') . '/';
    // Prüfen ob der absolute Pfad und angefordete Datei übereinstimmen.
    // Hier mit strpos und typgenauer(!) Prüfung auf 0:
    if ( ! 0 === strpos($getFileTainted, $thisDir) ) {
        show_forbidden ();
        exit;
    } else {
    // Nach dieser Prüfung gilt der Dateiname als sauber:
        $getFile = $getFileTainted;
    }

    //// Wenn es ein Verzeichnis ist:
    if ( is_dir($getFile) ){
        show_forbidden ();
        exit;
    }
```

```
    //// Eigentliches Ausliefern
    if ( is_readable($getFile) ) {
        if ( function_exists('finfo_open') ) {
            $finfo    = finfo_open(FILEINFO_MIME_TYPE);
            $mimeType = finfo_file($finfo, $getFile);
        } elseif ( function_exists('mime_content_type') ) {
            $mimeType = mime_content_type($getFile);
        } else {
            $mimeType = "application/unknown";
        }

        header('Content-Type:' . $mimeType);
        readfile($getFile);
        exit;

    //// Abbruchbedingung erfüllt:
    // Datei nicht lesbar
    } else {
        show_forbidden ();
        exit;
    }
} else {
    //// Abbruchbedingung erfüllt:
    // $_GET['file'] war leer oder nicht gesetzt:
    show_not_found ();
    exit;
}
```

Hinweis zu einer häufigen Fehlerquelle

Fehlerbild

- Nicht angemeldete Benutzer sehen nur eine (scheinbar) leere Seite und nicht das Anmelde-Formular und/oder
- Benutzer sehen nach dem Abmelden nur eine (scheinbar) leere Seite und /oder
- Benutzer sehen Fehlermeldungen, die mit `"Warning: Cannot modify header information"` beginnen.

Abhilfe und Vermeidung

Achten Sie bei der Einbindung darauf, dass vor `<?php require_once('auth.php'); ?>` wirklich nichts steht und achten Sie eben so sorgfältig darauf, dass vor dem "PHP-Tag" `<?php` in den Dateien `auth.php`, `login.php` sowie `logout.php` wirklich nichts steht. Benutzer mancher Editoren, vor allem des mit Windows mitgelieferten Notepad haben hier häufig ein Problem: Dieser speichert gemäß seiner Voreinstellung bei Texten mit der Zeichenkodierung UTF-8 ganz am Beginn der Datei zwei unsichtbare Bytes (genannt BOM - "byte order mark"), die also auch gesendet werden und die Funktion `header()` stören. Abhilfe schafft zwar, dieses im Menü zu unterbinden, zu empfehlen ist aber die Benutzung eines alternativen Editors wie z.B. Notepad++ - mit welchem Sie die BOM zum einen sehen und vor allem auch wieder löschen können...