Dieser Tipp beschreibt ein paar Methoden die HTTP-Authorization in PHP durchzuführen.
Es geht dabei aber nicht um eine vollständige und sichere Implementierung, sondern nur um die verschiedenen Techniken zum Zugriff.

Inhaltsverzeichnis

Warum?

Die HTTP-Authorization ist den meisten Benutzern und Programmieren durch `htaccess`/`htpasswd`-Dateien und die wenig schönen Dialoge bekannt, weshalb sich jeder Programmierer erstmal gleich ein HTML-Formular dafür baut.
HTTP-Authorization hat aber einen enormen Vorteil:
Es ist die einzige(1) Methode im Web ein Passwort sicher zu übertragen.
Das gilt allerdings nur für die `Digest`-Methode (und auf **keinen Fall** für die `Basic`-Methode), die von jedem modernen Browser (dazu gehört der Internet Explorer 6 nicht!) unterstützt wird.

Was ist mit SSL?

HTTPS wird gerne als Lösung für das Problem der unsicheren Übertragung von Passwörtern angeführt, ist es aber nicht!
Auch im SSL-Tunnel sind die Passwörter selbst genauso unverschlüsselt und mit dem Aufkommen von HTTPS-Inspection in Firmen(2) damit genauso anfällig wie reguläres HTTP.
HTTPS ist nur dann als sicher zu bezeichnen, wenn man seinen kompletten Zertifikat-Store löscht und jedes Zertifikat einzeln per Fingerprint hinzufügt!

Ansätze

Im folgenden gibt es ein paar Ansätze, wie man HTTP-Authorization verwenden kann und meine zwei Cent dazu, was ich davon halte

htpasswd-Dateien

Die simpelste Methode ist es tatsächlich die `.htusers`-Dateien von Apache zu verwenden. Über das `htdigest`-CLI-Programm und einen entsprechenden Aufruf in PHP liese sich die Verwaltung der Benutzer steuern.
Der angemeldete Benutzer steht dann in PHP zur Verfügung:
Quelltext | Drucken
echo $_SERVER['REMOTE_USER']; #=> "hans" echo $_SERVER['AUTH_TYPE']; #=> "Digest"
01.

```
echo $_SERVER['REMOTE_USER']; #=> "hans"
```

02.

```
echo $_SERVER['AUTH_TYPE']; #=> "Digest"
```

Wenn PHP auch als Apache-Modul läuft gibt es dabei auch noch die zusätzlichen, unten erwähnten Felder, so lange der Safe-Mode nicht aktiv ist.

In der Praxis sehe ich das nicht als Option, da ein Login-Feld ja erst erscheinen soll, wenn man wirklich auf "Login" klickt.
Mit `htaccess`-Dateien müsste dies aber schon vor dem ersten Aufruf der Seite geschehen (was somit auch Google ausschließt) oder die URLs für angemeldete Benutzer müssten abweichend definiert sein.
Darum werde ich hier auch auf die anderen Apache-Module (z.B. für LDAP) nicht weiter eingehen.

PHP als Apache-Modul

Wenn PHP als Apache-Modul läuft, ist es auch sehr einfach die Anmeldung zu behandeln.
Dazu muss man nur den Browser zuerst dazu bringen die Autorisierung einzufordern:
Quelltext | Drucken
$realm = "Meine Seite"; header('HTTP/1.1 401 Unauthorized'); header('WWW-Authenticate: Digest realm="' . $realm .'",qop="auth",nonce="' . uniqid() . '",opaque="' . md5($realm) .'"');
01.

```
$realm = "Meine Seite";
```

02.

```
header('HTTP/1.1 401 Unauthorized');
```

03.

```
header('WWW-Authenticate: Digest realm="' . $realm .'",qop="auth",nonce="' .
uniqid() . '",opaque="' . md5($realm) .'"');
```

PHP verarbeitet dann die Antwort-Daten des Clients und leitet diese im nächsten Durchlauf an das Script weiter:
Quelltext | Drucken

echo $_SERVER['PHP_AUTH_USER']; #=> "hans" echo $_SERVER['PHP_AUTH_DIGEST']; #=> username="hans", realm="Meine Seite", nonce="...", uri="...", response="...", opaque="...", cnonce="...", nc=00000001, qop="auth"

01.
```
echo $_SERVER['PHP_AUTH_USER']; #=> "hans"
```

02.
```
echo $_SERVER['PHP_AUTH_DIGEST']; #=> username="hans", realm="Meine Seite", nonce="...", uri="...", response="...", opaque="...", cnonce="...", nc=00000001, qop="auth"
```

Das Parsen des Strings ist dem Programmierer selbst überlassen.
Ein genaueres Beispiel (und das Beispiel für `Basic`) findet sich unter
http://php.net/manual/en/features.http-auth.php

Das Problem dieser Methode ist, dass man sich hiermit auf genau eine Konfiguration einschränkt, aber PHP grade im Shared-Hosting oftmals als (F)CGI läuft, und dieser Trick nicht funktioniert, deshalb würde ich davon eher Abstand nehmen.

apache_request_headers()

Eine andere Methode - wenn PHP als Apache-Modul läuft - ist die `apache_request_headers()`-Funktion, die alle Header als assoziatives Array zurückliefert.

Quelltext | Drucken

$header = apache_request_headers(); echo $header['Authorization']; #=> ...

01.
```
$header = apache_request_headers();
```

02.
```
echo $header['Authorization']; #=> ...
```

Der Kommentar zu dieser Methode ist der selbe wie oben

PHP mit FastCGI

Läuft PHP über `mod_fastcgi` lässt sich über die Apache-Konfigurations-Option

Quelltext | Drucken

FastCgiConfig -pass-header Authorization

01.
```
FastCgiConfig -pass-header Authorization
```

FCGI dazu bewegen den Header weiterzugeben.

Quelltext | Drucken

echo $_SERVER['Authorization']; #=> ...

01.
```
echo $_SERVER['Authorization']; #=> ...
```

Der Rest verhält sich dann wie oben.
Siehe: http://www.fastcgi.com/mod_fastcgi/docs/mod_fastcgi.html#FastCgiConfig

Wenn man allerdings `mod_fcgid` benutzt, lautet der Eintrag
Quelltext | Drucken
FcgidPassHeader Authorization
01.

```
FcgidPassHeader Authorization
```

(Unter älteren Versionen heißt die Direktive noch `PassHeader`) und die PHP-Variable heißt wieder `HTTP_AUTHORIZATION`.
Siehe: http://httpd.apache.org/mod_fcgid/mod/mod_fcgid.html#fcgidpassheader

Diese Methode ist somit eigentlich nur ein Workaround für Apache-Server.

द्मod_rewrite

Der Kreativität geschuldet ist die folgende Variante (von der man auch einige Abwandlungen findet), die Apaches `mod_rewrite` benutzt um den `Authorization` Antwort-Header an das PHP-Skript weiterzugeben:
Quelltext | Drucken
RewriteEngine on RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]
01.

```
RewriteEngine on
```

02.

```
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization},L]
```

Dieser Ausdruck trifft auf jede URL zu (`.*`), ohne sie zu verändern (`-`) und hängt den zusätzlichen Wert an. Außerdem ist das die letzte Rewrite-Regel, die ausgeführt wird (`L`).
In PHP selbst kann man dann die selbe Variable wie oben benutzen, um auf die Daten zugreifen zu können, wobei einige Apache-Versionen noch ein `REDIRECT_` vor die Variable hängen.

Ob man das als eine saubere Lösung betrachtet, möchte ich mal jedem selbst überlassen

द्मApache und der Authorization-Header

Das eigentliche Problem aller obigen Techniken ist, dass Apache sich weigert den `Authorization`-Header an CGI-Skripte weiterzugeben. Die Begründung dabei ist:

- CGI-Skripts könnten Zugriff auf Anmelde-Informationen erhalten (was ja hier genau beabsichtigt ist)
- Ein lokaler Systembenutzer könnte mit `ps e <PID>` die Benutzerdaten auslesen (was wir auf einem Server in Kauf nehmen können, da nur root dieses Recht besitzt)

Die Diskussion dazu findet sich in einem Bugreport von 1997: http://archive.apache.org/gnats/549

Man kann Apache aber auch dazu bringen diesen Header weiterzugeben, indem man die Compiler-Flag `-DSECURITY_HOLE_PASS_AUTHORIZATION` setzt.
Quelltext | Drucken
export CFLAGS="-DSECURITY_HOLE_PASS_AUTHORIZATION $CFLAGS" ./configure [...] make
01.

```
export CFLAGS="-DSECURITY_HOLE_PASS_AUTHORIZATION $CFLAGS"
```

02.

```
./configure [...]
```

03.

```
make
```

Diese Methode ist (aus der Natur der Sache) natürlich nur für Leute mit eigenem (v)Server möglich, die sich ihren Apache selbst kompilieren.

दFazit

Wer in einem PHP-Skript HTTP-Authorization benutzen will und dabei möglichst viele PHP- und Server-Kombinationen unterstützen will hat schlicht Pech gehabt: Es gibt keine Lösung, die auf jedem Server funktioniert, dafür allein schon 4 nur für Apache.
Wenn man diese Methode benutzen will, bleibt also nur das Arbeiten mit Weichen oder eine strikt definierte Umgebung.

1: Javascript bietet auch die Möglichkeit ein Passwort vor dem Übertragen zu hashen, aber da es nicht zwangsweise aktiviert ist, lasse ich es hier mal unter den Tisch fallen
2: Nur zur Erinnerung: Jeder Windows- und Mac-Rechner traut standardmäßig der "Deutschen Telekom Root CA" und Firefox traut gerne Zertifikaten, die niemandem gehören wollen.