Restricting Access

User authentication, passwords

User Authentication

- Nowadays most internet applications are available only for registered (paying) users
- How do we restrict access to our website only to privileged users?
- Use login forms for user authentication

A simple login script


```
<html>
<head><title>User Authentication</title></head>
<body>
<?php
$user = strtolower($_POST["user"]);
$pass = strtolower($_POST["pass"]);
if (isset($user) && isset($pass) && $user=="php5" && $pass=="iscool") {
?>

<h1>Welcome! Here is the truth about aliens visiting Earth ...</h1>
<?php
   } else {
?>

<h3>Please login</h3>
<form method=post>
   User name: <input type=text name=user /><br/>
   Password: <input type=password name=pass /><br />
<input type=submit name=submit value=Login />
</form>
<?php
   }
?>
</body>
</html>
```

What are the limitations of this?

Limitations of simple login script

- It only protects the page on which it is included.
- We could include it on all pages we wish to protect using something like:

```
include "login.php"
```

- Clearly not a good solution!

include

vars.php

```
<?php

$color = 'green';
$fruit = 'apple';

?>
```

When a file is included, the code it contains **inherits the variable scope** of the line on which the include occurs.

test.php

```
<?php

echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

PHP Sessions

- Create a true login “session”
- Use session variables to tag a “logged in” status

Login script using sessions


```php
<?php
session_start();
if (isset( $_POST["submit"] ) )// Have the credentials been submitted?
{
  $user = strtolower($_POST["user"]);
  $pass = strtolower($_POST["pass"]);
  if ($user=="dug" && $pass=="paradise") {
       $_SESSION["username"] = $user;
  }
  else {
    echo "<p>Login incorrect!";
  }
}   ?>

<html><head><title>Authentication</title></head><body>
<?php
if (isset( $_SESSION["username"] ) ) {
    echo "<p>You are logged in";
} else {
?>
<h3>Please login</h3>
<form method=post>
User name: <input type=text name=user /><br/>
Password: <input type=password name=pass /><br />
<input type=submit name=submit value=Login />
</form>
<?php   }  ?>
```

More Advanced logins

- Redirect after logging in
- Store login information for registered users
  - Take login information from a file storing usernames and passwords of registered users
  - Store this information in a database (see later)
  - Raises issues of data security.

Authentication using Apache

Part 1: Server Configuration: httpd.conf

Apache

- **Authentication:**
  - Verify if the user/passwd correct
- **Authorization:**
  - Once authenticated, does user have permission?
- **Access control**
  - Grant or deny access based on some criteria
  - e.g., **IP address, group name, domain** ...

File-based authentication

- **Apache** has a module that provides authentication
  - **mod_auth_basic**
  - Similar to /etc/passwd in Unix
  - Entries look like **admin**:kajsJjkh97U (encrypted)
- To add a user (Linux and Windows)
  - **htpasswd -c <file> <userid>**
    - This creates a new file with encrypted passwords
  - **htpasswd <file> <userid2>**
    - Appends other users

Creating a user password file

SYNTAX: htpasswd -c <file> <userid>

```
C:\Program Files (x86)\EasyPHP-5.3.3\apache\bin>htpasswd -c password napoleon

Automatically using MD5 format.
New password: ********
Re-type new password: ********
Adding password for user napoleon
```

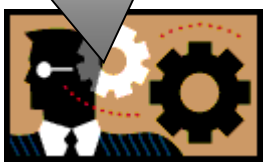
Create a new file (will delete if it exists already)

To create the file, use the **htpasswd** utility that came with Apache. This is located in the **bin** directory of wherever you installed **Apache**.

Filename: **password**

```
napoleon:$apr1$B5gzgGKw$AWVqtO2Romn5B4Zkc1bPk0
```

Encrypted password = hash key or message digest
- One way



Settings for this Example

Note: The password file was moved to another directory.

Password File:

C:\Program Files (x86)\EasyPHP-5.3.3\apache\users**password**

Restricted Directory:

C:\Program Files (x86)\EasyPHP-5.3.3\www**protected**

Configuring Apache: httpd.conf

Filename: **httpd.conf**

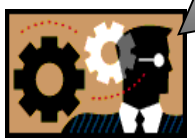
```
<Directory  "${path}/www/protected">
      AuthUserFile  "${path}/apache/users/password"
      AuthName        "This is a protected area"
      AuthGroupFile  /dev/null
      AuthType     Basic
      Require        valid-user
</Directory>
```

Directory you want to restrict access to

File containing user passwords

${path} = directory of **easyPHP**

Putting authentication directives in a **<Directory>** section, in your main server configuration file, is the preferred way



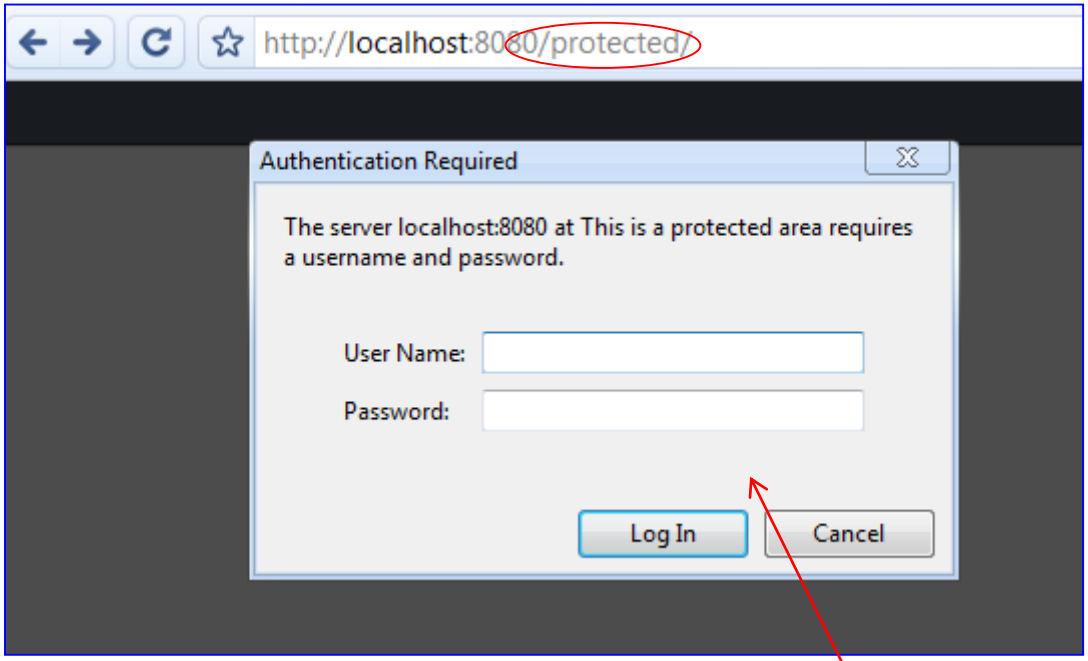
Configuration Parameters

AuthUserFile	The location of the password file.
AuthName	The authentication realm or name. This is the message that the user will see in the username/password pop-up.
AuthGroupFile	The location of the group file, if any.
AuthType	The type of authentication being used.
Require	What conditions need to be satisfied in order to allow the user through. It could be more than one condition.

Summary of Steps

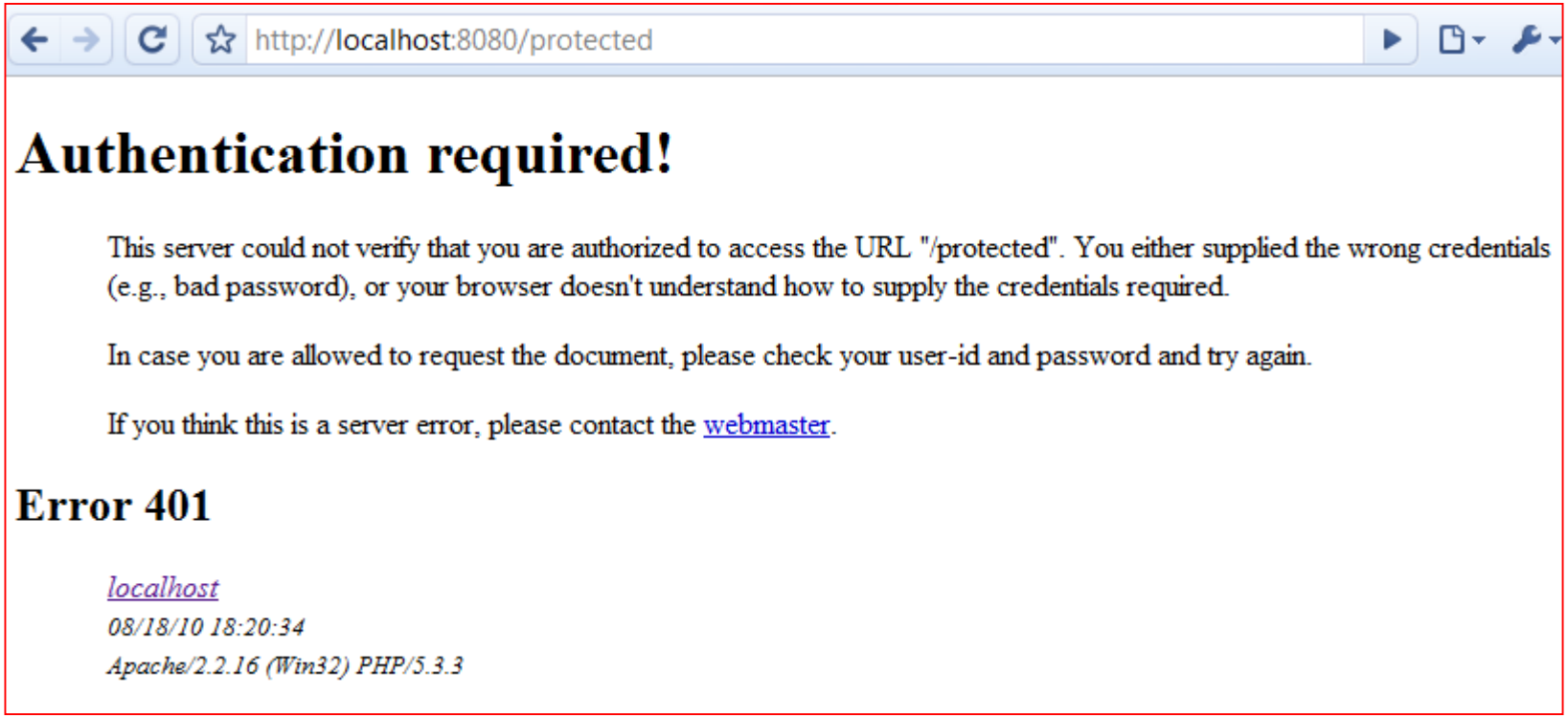
1. Create a password file.
2. Move the password file into a separate folder.
3. Create a directory to be restricted access to.
4. Modify Apache's **httpd.conf.**
5. Restart Apache webserver after making the modifications.

Sample Run: Accessing a "protected" section of your site



a dialog box automatically pops-up for user authentication

Sample Run: Invalid user name, password!



Demo

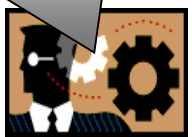
- See Apache configuration, using EasyPHP.
- Find mod_auth_basic

Authentication using Apache

Part 2: .htaccess files

The .htaccess file is used to **override** default server settings in particular folders (directories).

.htaccess files should be used **only** if you don't have access to the main server configuration file



File-based authentication

- By default overriding is not set on
- httpd.conf (Linux: in /etc/httpd/conf/)

```
...
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
...
```

Specifies which directives declared in the .htaccess file can override earlier configuration directives.

Follow Symbolic Links

If FollowSymlinks is NOT set at all, Apache has to issue some extra system calls when looking for a file.

For example, if you browse to the **/index.html** document, Apache would look for that file in your **/www, /www/htdocs**, and **/www/htdocs/index.html**.

These additional system calls will add to the latency. The system call results are not cached, so they will occur on every request.

http://www.maxi-pedia.com/FollowSymLinks

Example: File-based authentication

Filename: **httpd.conf**

```
AccessFileName .htaccess

 <Directory />
    Options FollowSymLinks
    AllowOverride None
    Order Deny, Allow
    Deny from All
 </Directory>

 <Directory "${path}/www/protected2">
     AllowOverride All
     Order allow,deny
     Allow from all
 </Directory>
```

Note that the default Apache access for <Directory /> is Allow from All. This means that Apache will serve any file mapped from an URL.

This is the recommended initial setting!

We can then override this for directories we want accessible.

http://httpd.apache.org/docs/2.2/mod/core.html#directory

Specifies which directives declared in the .htaccess file can override earlier configuration directives.

Settings for this Example

Note: The password file was moved to another directory.

Away from the folder open to the public (not in the document root)

Password File:

C:\Program Files (x86)\EasyPHP-5.3.3\apache\users**password**

Restricted Directory:

C:\Program Files (x86)\EasyPHP-5.3.3\www**protected2**

Example: .htaccess

Filename: **.htaccess**

```
AuthUserFile  "C:/Program Files/EasyPHP-5.3.2i/apache/users/password"
AuthName      "Protected Area 2"
AuthGroupFile  /dev/null
AuthType      Basic
Require       valid-user
```

Create the file using a temporary name first, then rename it afterwards

implemented by **mod_auth_basic. (Alternatively, mod_auth_digest for Digest), better but non-standard yet.**

To implement authentication, you must also use the **AuthName** and **Require** directives. In addition, the server must have an authentication-provider module such as **mod_authn_file** and an authorization module such as **mod_authz_user.**

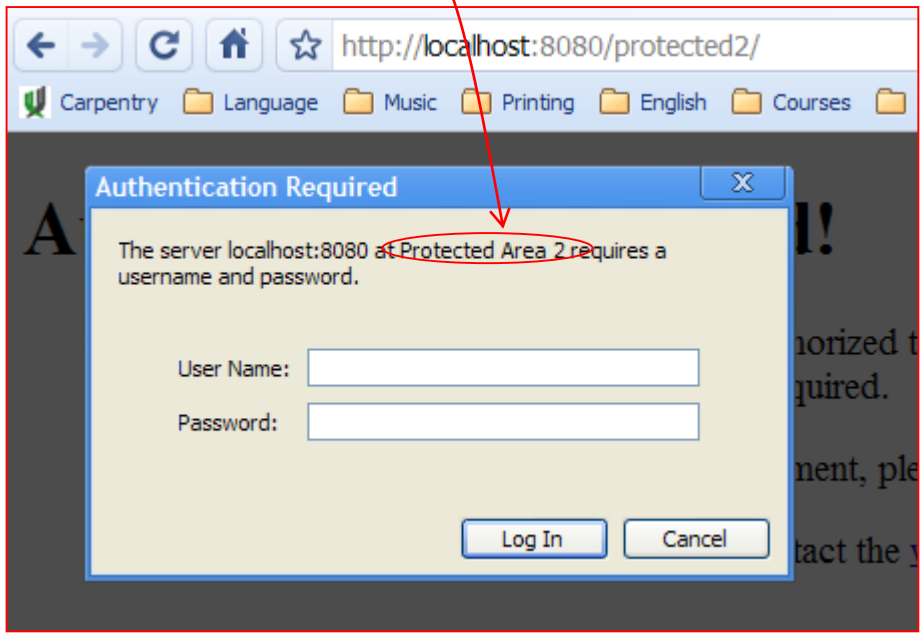
Directory: **C:\Program Files\EasyPHP-5.3.2i\www\protected2**

Your **.htaccess** file should reside in this **directory**

Sample Run

Filename: .htaccess

```
AuthUserFile  "C:/Program Files/EasyPHP-5.3.2i/apache/users/password"
AuthName      "Protected Area 2"
AuthGroupFile  /dev/null
AuthType    Basic
Require       valid-user
```



.htaccess files

- **Pros**
  - Easy way to allow authentication
  - Control is given to users (**developers**)
  - No admin
- **Cons**
  - Performance: Apache reads and looks for **.htaccess** files for every GET
  - Wrong permissions (given by directory owners) can lead to security problems

Another .htaccess Example (Linux)

- If allowed, the following **.htaccess file** overrides authorisation:

```
<Directory /www/mysite/example>
    IndexIgnore * ##does not allow dir lists
    AuthType Basic
    AuthName "Private Area" ##popup
    AuthUserFile /usr/local/apache/passfile
    AuthAuthoritative on
    Require valid-user
</Directory>
```

More on Apache Webserver's Configuration

Options **FollowSymLinks**

Order **Allow, Deny**

Options FollowSymLinks

Follow Symbolic Links

- Websites are often set up in a way that they show pictures and other content as being physically located at some other location than they really are.

- If a visitor requests /system/files/images/image.png

  then show him /pictures/image.png.“

- You might see something like

  IMG SRC="/system/files/images/image.png" for the location of some picture. This would be viewable by a browser, but not downloadable as it resides in another physical directory.

- **Enable** FollowSymLinks by default

Order Allow, Deny

Restricting access

• **Order allow, deny** is a setting in your Apache web server configuration that is used to **restrict** access to **certain directories** (folders) or even **globally**.

• Configuring who can access your directories is very important for your web site security.

• Order allow,deny is one way to restrict who can see what.

Order Allow, Deny

Restricting access

- The **Allow** directive affects which hosts "can access" an area of the server. Access is usually controlled by **hostname, IP address, or IP address range**.

- The **Deny** directive "restricts access" to the server. Restrictions can be based again on **hostname, IP address, or environment variables**.

- We can set the Order directive in two ways:
  - **Order allow, deny**
  - **Order deny, allow**

Order Allow, Deny

Restricting access

- **Order allow, deny** tells your web server that the *Allow* rules are processed before the *Deny* rules.
- If the client does not match the *Allow* rule or it does match the *Deny* rule, then the client will be denied access.

- **Order deny, allow** means that the deny rules are processed before the allow rules.
- If the client does not match the deny rule or it does match the allow rule, then it will be granted access.

Order Allow, Deny

Example of Allow

Allow from example.com

- All hosts from this domain will be allowed.
- Allowed: **abc.example.com** ✓

  www.example.com. ✓

Not Allowed: **www.abcexample.com** ✗

- Only complete components are matched

Order Allow, Deny

Example of Allow

Allow from example.com

- This configuration will cause the server to perform a double reverse DNS lookup on the client **IP address**, regardless of the setting of the **HostnameLookups** directive.
- It will do a **reverse DNS lookup** on the **IP address** to find the associated **hostname**, and then do a **forward** lookup on the **hostname** to assure that it matches the original **IP address**.
- Only if the **forward and reverse DNS** are consistent and the hostname matches will access be allowed.

http://httpd.apache.org/docs/2.0/mod/core.html#hostnamelookups

Order Allow, Deny

Example of Allow

Allow from 10.1.2.3

- You can define the access level also by providing the IP address. In this example, just the host with just that IP address would be allowed access.

Allow from 10.1

- All hosts from all **subnets** within **10.1.x.x** would be allowed access.

Order Allow, Deny

Example

```
<Directory "/www">
    Order Allow, Deny
    Deny from all
    Allow from all
</Directory>
```

- In this case, your **client** would be **denied access**. Why?

- Apache first evaluates the Allow directive rules and then the Deny directive rules.

- Allow from all would be executed first and then the Deny from all would take place.

Order Deny, Allow

Example: order has been swapped

```
<Directory "/www">
    Order Deny, Allow
    Deny from all
    Allow from all
</Directory>
```

• The configuration above would result in your **client** being **allowed access** because the Deny from all rule would be processed first and the Allow from all rule would be processed second.

Order Deny, Allow

Example: restricted server, intranet site

```
<Directory "/www">
    Order Deny, Allow
    Deny from all
    Allow from example.com
</Directory>
```

- This configuration would **restrict** **everyone** from accessing the **/www** directory **except** hosts in the **example.com** domain.
- **Abc.example.com** would be allowed access ✓
- **www.myexample.com** would be restricted. ✗
- Only complete components are matched

Order Allow, Deny

Example: blocking someone from some specific domain

```
<Directory "/www">
    Order Allow, Deny
    Allow from all
    Deny from www.myattacker.com  phishers.example.com
</Directory>
```

- The configuration provided above would give access to everyone and **restrict** **all hosts** from the **www.myattacker.com** and **phishers.example.com** domains.

Order Allow, Deny

What happens if you forget to provide specific rules and use just the Order allow,deny directive alone?

```
<Directory "/www">
    Order Allow, Deny
</Directory>
```

- when you specify the Order allow,deny you also control the **default access state**.

- The example above will **Deny <u>all access</u>** to the **/www** directory because the default access state is set to Deny.

IndexIgnore

IndexIgnore relates to the **default directory listing mechanism** that returns a directory listing for directories which do not contain an **index.html** or other "index" file. If that file is present, then IndexIgnore does not do anything.

IndexIgnore file [file] ...

- You can find the IndexIgnore directive in two places.
  - **httpd.conf** Apache server configuration file
  - **.htaccess** file
    - If you edit IndexIgnore in your root .htaccess file, it will affect all subdirectories as well. If you want to apply your setting to a subdirectory only, then you have to add a .htaccess file to that subdirectory and edit that.

- IndexIgnore relies on the **mod_autoindex** module. Without this module enabled, no directory listings take place.

IndexIgnore

Examples

IndexIgnore readme.txt .htaccess

• Disable the **readme.txt** and **.htaccess** files from showing in your directory listing.

IndexIgnore *

• Block the directory listing completely

Summary

- Methods for user authentication
  - Simple login scripts
  - HTTP authentication
  - Authentication using sessions
- Enable **FollowSymLinks** by default

Summary

- The **Allow** and **Deny** directives are used to specify which clients are or are not allowed access to the server.
- The **Order** directive sets the **default access state**, and configures how the Allow and Deny directives interact with each other.