

PHP Authentication Schemes

IN THIS CHAPTER

- Overview
- Generating Passwords
- Authenticating User Against Text Files
- Authenticating Users by IP Address
- Authenticating Users Using HTTP Authentication
- Authenticating Users by Database Query

User Authentication Overview

If you've ever built a Web site that requires user registration, then you have some idea of what is required to restrict access so that you can be sure the user who is registered is the same user who is currently accessing the site. In almost all cases, you must require the user to enter some sort of login and password to gain access to the site.

The current trend in site registration seems to be to require the users to enter their email as part of the registration process, and then subsequently send the users' password to them using that email address. This provides a number of benefits. First, you have the user's email and you know it is at least semivalid,

since the user cannot gain access to the site without first submitting an email address and being sent a password. If users forget their passwords, you can easily send it to them (or a hint, if you require one) using some simple code. Additionally, you are able to send updates to users about site features, special events, etc., and you know the updates are going to valid addresses. On a side note, if you do send email to users, you should always ask their permission first in an open “opt-in” method. Additionally, you should provide an automated way for users to unsubscribe by clicking a link in the email you do send. Spam is a problem; let’s not contribute to this ever-worsening problem.

Another advantage of requiring an email address is the ability to use it as the user’s login. Email addresses must be unique, and only the valid holder of the email is able to receive any passwords that you send. This solves the problem of unique logins for every user. You do not have to write error-checking into your code that suggests alternative login names should one already be taken.

These authentication methods are for casual sites. They should not be used to protect sensitive information, such as social security numbers, credit card numbers, or anything that you wouldn’t leave sitting on the street corner. There are ways to get around these methods, be it from hacking, IP spoofing, or good old brute force. However, these methods do work well for community sites where basic authentication is required and there is little to gain from spending hours upon hours in cracking attempts. A malicious hacker has lots to gain when credit cards are on the line, but when the reward is only a user’s list of favorite links, there is little incentive to spend the effort required to crack a system. Simply put, don’t assume that your information is safe by placing it solely behind a PHP authentication solution.

This chapter goes in depth into some of the above scenarios, as well as some additional ones, such as restricting logins to one domain or even a range of IP addresses.

Generating Passwords

The `md5()` and `crypt()` functions encrypt passwords, but they cannot be unencrypted. These are one-way algorithms. You can verify that the users' password matches the password they were initially given by comparing the `md5()` or `crypt()` output of the password they use to subsequently enter the site. The two encrypted versions of the same string match (assuming that the same "salt" is used to create the password using the `crypt()` function).

This is good, because you never store a user's actual password. If your password file falls into the wrong hands, there is little that anybody can do with it. It is very hard to unencrypt a password encrypted by `md5()` or `crypt()`. Since you don't store the user's actual password, malicious hackers who may get their hands on your password file can't take that password and easily use it to attempt to break into other sites that your user may visit, since, unfortunately, most people don't use a different password for every site they visit.

Later scripts in this chapter assume that you have already created some sort of file containing usernames and passwords. The general convention for storing passwords in text files is to put one username/password combination on each line, and to separate the user and password with a colon. For example:

```
user1:sih2hDulacVcA
user2:aSP2C8UUWnxjA
```

The first script in this chapter creates an `md5()` encrypted password and a `crypt()` encrypted password for any string you enter. As shown in Figure 7-1, you can use this script to easily generate encrypted passwords and display them on the screen so that you can copy and paste them into a text file. The `crypt()` encrypted password generated from the script is the same as encrypting a password using Apache's `htpasswd` program.

SCRIPT 7-1 `generating_passwords.php`

1. `<html>`
2. `<head>`
3. `<title>Password Creator</title>`

SCRIPT 7-1 generating_passwords.php (Continued)

```

4. </head>
5. <body>
6. <form action=generate_passwords.php method=post>
7. <h3>Enter a password to create MD5 and Crypt based
   passwords.</h3>
8. Password: <input type="text" name="password">
9. <input type="submit" name="create" value="Create
   Passwords!">
10. </form>
11. <?
12. if(isset($password)) {
13.     ?>
14.     <h3>The passwords for the string "<?=$password?>"
       are:</h3>
15.     <ul>
16.     <li><b>MD5:</b> <?=md5($password)?>
17.     <li><b>Crypt:</b> <?=crypt($password)?>
18.     </ul>
19.     <?
20. }
21. ?>
22. </body>
23. </html>

```

Script 7-1 generating_passwords.php Line-by-Line Explanation

| LINE | DESCRIPTION |
|-------|---|
| 1-10 | Create an HTML form with one text input field, named "password," and a submit button. |
| 11 | Start parsing the page as PHP. |
| 12 | Check to see if the \$password variable has been set. If it has, continue to line 13; if not, continue on line 20. |
| 13-19 | Stop parsing the page as PHP. Print out the values of the password after it has been encrypted by the md5() and crypt() functions. Start parsing the page as PHP again. |
| 20 | End the if statement started on line 12. |
| 21 | Stop parsing the page as PHP. |
| 23 | Print out the closing HTML for the page. |



FIGURE 7-1 `generating_passwords.php`

Authenticating Users Against Text Files

Not all sites contain a database back-end to store data, and some sites probably will never require them. Still, you may want to limit access to certain parts of the site or even the entire site. One way to do this is to authenticate against a text file stored someplace on the server, preferably out of the Web server directory so that there is no way it can be accessed by someone with a Web browser.

This method only requires the standard file functions, as well as the MD5 encryption function.

This script serves a dual purpose. It allows you to add users to a password file, as well as test authentication. This way, you can create a blank text file and easily fill it with some username/password combinations so that you can test the functionality of the script.

This script is not meant to be used in its entirety to check passwords on a site. You would need to remove the functions that allow you to add users before you placed the script in your own site.

SCRIPT 7-2 file_authentication.php

```
1. <?
2. $password_file = "C:/apache/passwords/pass.txt";
3.
4. function check_pass($login, $password) {
5.     global $password_file;
6.     if(!$fh = fopen($password_file, "r")) {die("<P>Could
Not Open Password File");}
7.     $match = 0;
8.     $password = md5($password);
9.     while(!feof($fh)) {
10.         $line = fgets($fh, 4096);
11.         $user_pass = explode(":", $line);
12.         if($user_pass[0] == $login) {
13.             if(rtrim($user_pass[1]) == $password) {
14.                 $match = 1;
15.                 break;
16.             }
17.         }
18.     }
19.     if($match) {
20.         return 1;
21.     } else {
22.         return 0;
23.     }
24.     fclose($fh);
25. }
26.
27. function print_login_form($login) {
28.     ?>
29.     <p>Please Log In:
30.     <form action=file_authentication.php method=post>
31.     <p>Login: <input type="text" name="login"
value="<?=$login?>">
32.     <br>Password: <input type="password" name="password">
33.     <br><input type="submit" name="checkpass" value="Login!">
34.     </form>
35.     <?
36. }
37.
38. function print_add_form() {
39.     ?>
40.     <p>Add New User:
41.     <form action=file_authentication.php method=post>
42.     <p>Login: <input type="text" name="adduser">
43.     <br>Password: <input type="password" name="addpass">
```

SCRIPT 7-2 file_authentication.php (Continued)

```
44.     <br><input type="submit" name="add" value="Add User!">
45. </form>
46. <?
47. }
48.
49. function add_user($adduser, $addpass) {
50.     global $password_file;
51.     if(!$fh = fopen($password_file, "a+")) { die("<P>Could
Not Open Password File"); }
52.     rewind($fh);
53.     while(!feof($fh)) {
54.         $line = fgets($fh, 4096);
55.         $user_pass = explode(":", $line);
56.         if($user_pass[0] == $adduser) {
57.             echo "<h2>Duplicate Login. Invalid!</h2>";
58.             return 0;
59.         }
60.     }
61.     $add = $adduser . ":" . md5($addpass) . "\n";
62.     if(!fwrite($fh, $add)) { die("<P>Could Not Write To
Password File"); }
63.     fclose($fh);
64.     echo"<h2>User Added!</h2>";
65. }
66. /***** MAIN *****/
67. if(isset($checkpass)) {
68.     if(check_pass($login, $password)) {
69.         echo "<h2>Login Success!</h2>";
70.     } else {
71.         echo "<h2>Login Failed</h2><p>Bad username or
password. Login and Password are case-sensitive. Try
again:";
72.         print_login_form($login);
73.     }
74. } elseif(isset($add_form)) {
75.     print_add_form();
76. } elseif(isset($add)) {
77.     add_user($adduser, $addpass);
78. } else {
79.     print_login_form("");
80. }
81. ?>
82. <p>You can <a href=file_authentication.php?add_form=1>Add
Users</a> or <a href=file_authentication.php?login</a> an
existing user.
```

Script 7–2 file_authentication.php Line-by-Line Explanation

| LINE | DESCRIPTION |
|------|--|
| 2 | Define the file that contains the passwords. Note that you should specify the correct path on your machine that points to the password file. On *nix-based systems, you'd obviously omit the "C:\\" nonsense in the path. |
| 4 | Create a function that checks the user-entered password against the username/password combination stored in the password file. |
| 5 | Allow the password file to be read and modified globally by this function. |
| 6 | Attempt to open the file and assign it to a file pointer. If the file cannot be opened, then print an error and exit from the script. |
| 7 | Initialize a variable to track a username/password match. Initialize it to false (0) to assume the password is incorrect. |
| 8 | Since the passwords are stored encrypted, encrypt the user-entered password using the MD5 function. |
| 9–18 | Create a while loop that searches through the file until the end of file (EOF) is reached. |
| 10 | For each iteration of the loop, get one line from the file and place it in the \$line variable. |
| 11 | Create an array of the line obtained from the file using the explode function. Since the usernames and passwords are stored in the format: USER:PASS and the explode() function breaks up the line on every ":", the script places two items into the \$user_pass array: the username at index [0] and the password at index [1]. |
| 12 | Check if the username from the file, which is stored in \$user_pass[0], matches the username entered by the user, \$login. |
| 13 | If the user-entered username matches the username obtained from the file, then check to see if the password from the file, \$user_pass[1], matches the password entered by the user, \$password. |
| 14 | If the passwords match, then set the \$match variable to true (1), since the user-entered username and password match the ones contained in the password file. |
| 15 | If the passwords match, then break from the loop. |
| 16 | Close out the password-checking if statement. |
| 17 | Close out the username-checking if statement. |
| 18 | Close out the while loop. |

Script 7-2 file_authentication.php Line-by-Line Explanation (Continued)

| LINE | DESCRIPTION |
|-------|--|
| 19-20 | If there was a match, then return true to the calling program or function. |
| 21-23 | If there was no match, then return false to the calling program or function. |
| 24 | Close the file. |
| 25 | End the function declaration. |
| 27 | Create a function that prints a login form to the screen. The function takes one argument, called <code>\$login</code> , which is used to automatically re-enter the username if there was no match from a previous attempt at logging in. |
| 28 | Stop parsing the page as PHP. |
| 29-30 | Begin displaying the login form. |
| 31 | Print the username field. If this is a subsequent attempt to log in, then automatically enter the username. |
| 32-34 | Continue printing the form to the page. |
| 35 | Start parsing the page as PHP again. |
| 36 | End the function declaration. |
| 38 | Create a function that prints a form allowing you to add username/password combinations to the password file. |
| 39-46 | Stop parsing the page as PHP and print out a standard HTML form, then continue parsing the page as PHP. |
| 47 | End the function declaration. |
| 49 | Create a function that adds a username and password combination to the password file. The function takes two arguments: <code>\$adduser</code> and <code>\$addpass</code> . |
| 50 | Allow the password file to be read and modified globally by this function. |
| 51 | Attempt to open the file and assign it to a file pointer. If the file cannot be opened, then print an error and exit from the script. |
| 52 | Since we opened the file using append mode ("a+") on line 51, rewind the file so that the file pointer is at the beginning of the file. The append mode places the file pointer at the end of the file by default. |
| 53-60 | Create a while loop that searches through the file until the end of file (EOF) is reached. |
| 54 | For each iteration of the loop, get one line from the file and place it in the <code>\$line</code> variable. |

Script 7–2 file_authentication.php Line-by-Line Explanation (Continued)

| LINE | DESCRIPTION |
|-------|---|
| 55 | Create an array of the line obtained from the file using the explode function. |
| 56 | Check to see if the username stored in <code>\$user_pass[0]</code> is the same as the username entered into the Add User form. Usernames should always be unique! This line checks to make sure that a duplicate username is not being entered into the password file. |
| 57 | If the usernames match, then you are trying to enter a duplicate username. Print out a message to the users informing them that the username they entered is invalid. |
| 58 | Return false to the calling program or function. |
| 61 | Since this line has been reached, it means that the username entered was unique (if it was not, then the function would have terminated with the return on line 58). Create a username/password combination by prepending the username to the MD5 encrypted form of the password. A colon is placed in between the two so that the script can parse easily when it checks usernames and passwords. Colons do not occur as output in MD5 encryption, so they are safe characters to use to delimit the two values. (Note that the script doesn't forbid colons in the usernames, which would cause user authentication to always fail if a user placed a colon in a username.) |
| 62 | Attempt to write the new username/password combination to the end of the file. You know you are at the end of the file, since the while loop just parsed the entire file. If the write attempt fails, then display an error and kill the script. |
| 63 | Close the file. |
| 64 | Print a message letting the user know the password was added to the password file. |
| 65 | End the function declaration. |
| 66 | Begin the main file execution. |
| 67 | Check to see if the <code>\$checkpass</code> variable has been set. If it is set, this means that the login form was submitted to the script. |
| 68 | If the <code>\$checkpass</code> variable is set, run the <code>check_pass()</code> function using the user-entered username and password. |
| 69 | If the <code>check_pass()</code> function returns true (1), notify the user that authentication succeeded. At this point, you would normally display the content that you were protecting with the user authentication. |
| 70–73 | If the <code>check_pass()</code> function returned false, then print a message informing the user of the error. |

Script 7-2 file_authentication.php Line-by-Line Explanation (Continued)

| LINE | DESCRIPTION |
|-------|--|
| 74-75 | If the <code>\$checkpass</code> variable is not set, but the <code>\$add_form</code> variable is set, then print out the form to allow you to add new username/password combinations to the password file. |
| 76-77 | If the <code>\$checkpass</code> variable is not set and the <code>\$add_form</code> variable is not set, but the <code>\$add</code> variable is set, then run the <code>add_user</code> function using the values entered into the Add User form. |
| 78-79 | If none of the above is true, then print the login form using a null value as the argument so that the <code>login_form</code> function does not generate a warning about use of an uninitialized variable. |
| 81 | Stop parsing the page as PHP. |
| 82 | Print a line to the browser that provides two links. The first is a link that sets the <code>add_form</code> variable so that the <code>add_form</code> function is called and you can add username/password combinations to the password file. The second link is used to display the login form after you have added a user. |

Authenticating Users by IP Address

In some rare instances, you may wish to limit access to a certain page or pages to certain IP addresses. It may be because you have an internal network that has a publicly viewable Web site. You may wish to have certain pages be viewable only by certain machines on the internal network.

Or it may be that you have your own remote Web site, and you wish to restrict access to a certain page or pages so that only you, from a static IP address, can access those pages.

These methods work by determining the IP address of the user trying to view the page and checking it against a set value. If the IP address matches, then the page can be viewed.

Note that this method could be fooled by IP spoofing or other hacks, and should not be used to protect sensitive information, such as credit card numbers, proprietary information, or the like. It is a simple method that can protect pages from the majority of users surfing the net.

When using this script, you should be aware that your computer may have more than one IP address, and that the one your browser is reporting to the script may not be the same one

you are attempting to verify against. This is especially true when the browser and Web server reside on the same machine, as you will probably encounter when testing the script.

For example, computers have a default local IP address of 127.0.0.1, otherwise known as “localhost.” When I start up the Web server and type in “http://localhost”, my browser is telling the script that I am at the IP address of 127.0.0.1. However, the Web browser on my computer is also running on the IP address 192.168.0.1 (an IP address used for an internal network). If I type in the IP address 192.168.0.1, my Web browser reports that I am at 192.168.0.1, but the script only accepts the IP address of 127.0.0.1! If you run into problems, try echoing the value of `$REMOTE_ADDR` to the screen to see which IP address your browser is reporting to the script.

SCRIPT 7-3 IP_authentication.php

```
1. <?
2. $accept = array ("127", "0", "0", "1");
3. $remote = explode(".", $REMOTE_ADDR);
4. $match = 1;
5. for($i = 0; $i < sizeof($accept); $i++) {
6.     if($remote[$i] != $accept[$i]) {
7.         $match = 0;
8.     }
9. }
10. if($match) {
11.     echo "<h2>Access Granted!</h2>";
12. } else {
13.     echo "<h2>Access Forbidden</h2>";
14. }
15. ?>
```

Script 7-3 IP_authentication.php Line-by-Line Explanation

| LINE | DESCRIPTION |
|-------|---|
| 2 | Define an array of the acceptable IP address(es) ranges that can view the page. The array must have at least one item. You can restrict by IP subnets by adding more or fewer items to the array. For example: array("192","168","0","1") limits the users who can view the page to the user with the IP address of 192.168.0.1. array("192","168","0") limits the users who can view the page to the users with an IP address in the range of 192.168.0.1 to 192.168.0.255. array("192","168") limits the users who can view the page to the users with an IP address in the range of 192.168.0.1 to 192.168.255.255. array("192") allows anybody with an IP that starts with 192 to view the page. Not very practical, but it would work! |
| 3 | Create an array of numbers based on the user's IP address by exploding the \$REMOTE_ADDR global variable on the periods in the IP address. |
| 4 | Define a variable, \$match, and set it to true (1). |
| 5-9 | Loop through the acceptable IP address array. |
| 6 | Compare the acceptable IP address segment with the IP address segment of the user's browser. |
| 7 | If the IP address segments do not match, then set the \$match variable to false (0). |
| 8 | Close the if statement. |
| 9 | Close the for loop. |
| 10-11 | If \$match is true, then allow access and print a message to the user. This is where you would present your protected content. |
| 12-14 | If \$match is false (0), then print out a message telling the users that they cannot access the page and exit from the script. |
| 15 | Close the if statement started on line 10. |

Authenticating Users Using HTTP Authentication

PHP has some built-in support for basic "HTTP Authentication." HTTP authentication is the type of authentication in which a small window pops up in front of the browser prompting you for a username and password. Figure 7-2 and Figure 7-3 show HTTP Authentication windows in Internet Explorer and Mozilla for Linux, respectively. HTTP authentication is server-dependent, the most common example of which is



FIGURE 7-2
HTTP Authentication Login
Window in Internet Explorer



FIGURE 7-3 HTTP Authentication Window in Mozilla on Linux

Apache's HTTP authentication method. The methods described here assume you are using an Apache server.

Typically, the HTTP Authentication method under Apache is defined in `.htaccess` files.¹ A `.htaccess` file usually restricts an entire directory. However, PHP provides a way to use Apache based HTTP Authentication without having to define `.htaccess` files for the same type of authentication. Additionally, you can easily use PHP's HTTP Authentication method on a per page basis.

Using the PHP method, you can also use your own password scheme. Under Apache, you typically placed encrypted passwords in a password file (usually called `.htpasswd`). Using the PHP method, you can place your passwords in a file, in a

1. Complete details are available at <http://httpd.apache.org/docs/howto/auth.html>.

database, or any other method. You can also use any encryption scheme you want, whereas Apache is restricted to UNIX crypted passwords, their own flavor of MD5 passwords, or plain text.

However, if users forget their passwords, you cannot send it to them, because you only know the encrypted version. You either have to reset the password or provide a hint for the password that may help the users remember what it is they used for the password. More on this later in the chapter.

The next script in this chapter details how to use PHP's method of HTTP Authentication with an Apache server. The script also allows you to use passwords encrypted with the crypt() function (which are portable and can be used with Apache's default HTTP Authentication method) or use the basic PHP md5() function to encrypt passwords. The latter option is useful if you already have several applications that use MD5 encryption for passwords.

SCRIPT 7-4 http_authentication.php

```
1.  <?
2.  function check_pass($login, $password, $mode) {
3.      global $password_file;
4.      if(!$fh = fopen($password_file, "r")) { die("<P>Could
      Not Open Password File"); }
5.      $match = 0;
6.      while(!feof($fh)) {
7.          $line = fgets($fh, 4096);
8.          $from_file = explode(":", $line);
9.          if($from_file[0] == $login) {
10.             if($mode == "crypt"){
11.                 $salt = substr($from_file[1],0,2);
12.                 $user_pass = crypt($password,$salt);
13.             } elseif ($mode == "md5") {
14.                 $user_pass = md5($password);
15.             }
16.             if(rtrim($from_file[1]) == $user_pass) {
17.                 $match = 1;
18.                 break;
19.             }
20.         }
21.     }
22.     if($match) {
```

SCRIPT 7-4 http_authentication.php (Continued)

```

23.     return 1;
24.   } else {
25.     return 0;
26.   }
27.   fclose($fh);
28. }
29. function authenticate() {
30.   Header("WWW-Authenticate: Basic realm=\"RESTRICTED
ACCESS\"");
31.   Header("HTTP/1.0 401 Unauthorized");
32.   echo("<h1>INVALID USERNAME OR PASSWORD. ACCESS
DENIED<h1>");
33.   exit;
34. }
35. /*** MAIN ***/
36. //select md5 or crypt for $mode. md5 is for md5 encoded
passwords, crypt is for passwords encoded using apache's
httpasswd
37. $mode = "md5";
38. $password_file = "C:/apache/passwords/pass.txt";
39. if (!isset($PHP_AUTH_USER)) {
40.   authenticate();
41. } else {
42.   if(check_pass($PHP_AUTH_USER, $PHP_AUTH_PW, $mode)) {
43.     ?>
44.     <h1>ACCEPTED</h1>
45.     <?
46.   } else {
47.     authenticate();
48.   }
49. }
50. ?>

```

Script 7-4 http_authentication.php Line-by-Line Explanation

| LINE | DESCRIPTION |
|------|--|
| 2 | Define a function called <code>check_pass()</code> that takes three arguments: <ul style="list-style-type: none"> • <code>\$login</code>—the user-entered username. • <code>\$password</code>—the user-entered password. • <code>\$mode</code>—the encryption mode used in the password file, “md5” or “crypt.” |
| 3 | Allow <code>\$password_file</code> to be used globally by this function. |

Script 7–4 http_authentication.php Line-by-Line Explanation (Continued)

| LINE | DESCRIPTION |
|-------|--|
| 4 | Attempt to open the file in read mode and assign a file handle to it. If the file cannot be opened, then print an error and kill the script. |
| 5 | Define the variable <code>\$match</code> and set it to false (0). |
| 6–21 | Create a for loop that loops through the file until the EOF is reached. |
| 7 | Read in one line from the file. |
| 8 | Explode the contents of the line into an array called <code>\$from_file</code> . This should create an array with two items. The first is the username, and the second is the password. |
| 9 | If the first element in the array (the username) matches the user-entered username (<code>\$login</code>), continue to line 10. Otherwise, go back to line 6 and loop again. |
| 10 | Check to see if the <code>\$mode</code> variable that was passed to this function is "crypt." If it is, continue to line 11. Otherwise, go to line 13. |
| 11 | Define the salt required by the <code>crypt()</code> function. This script can be used with Apache <code>htpasswd</code> generated passwords, so it has to use Apache's method of using <code>crypt()</code> s, which is to use the first two characters of the encrypted password as the salt. This line uses the <code>substr()</code> function to pull out the first two characters from the password obtained from the file and place the value in the <code>\$salt</code> variable. |
| 12 | Encrypt the user-entered password with the <code>crypt()</code> function using the salt obtained from the previous line. Continue to line 16. |
| 13 | Check to see if the <code>\$mode</code> variable that was passed to this function is "md5". If it is, continue to line 14. Note: The function assumes that <code>\$mode</code> is either "crypt" or "md5". If it is neither, then the script loops through the entire file without checking anything and returns no matches, which in turn means that authentication fails. It may be useful to print an error message that <code>\$mode</code> was not set to either of the required values, but that won't help a user who is trying to access your site. For debugging, you may want to optionally add another else statement that states that <code>\$mode</code> was neither "crypt" nor "md5". |
| 14 | Encrypt the user-entered password with the <code>md5()</code> function. Continue to line 16. |
| 16–17 | Trim any whitespace from the password obtained from the file and compare it to the encrypted version of the user-entered password. If they match, then set the <code>\$match</code> variable to true (1). |
| 18 | Break out of the loop, since you found a valid match. |

Script 7–4 http_authentication.php Line-by-Line Explanation (Continued)

| LINE | DESCRIPTION |
|-------|---|
| 19 | Close the if statement that checks the password. |
| 20 | Close the if statement that checks the username. |
| 21 | Close the for loop. |
| 22–23 | If <code>\$match</code> is true (1), then return true. |
| 24–26 | If <code>\$match</code> is false(0), then return false. |
| 27 | Close the file. |
| 28 | End the function declaration. |
| 29 | Create a function called <code>authenticate()</code> to display the HTTP Authentication login screen to users if they have not been previously authenticated. |
| 30–31 | Use the <code>header()</code> function to send authentication headers to a user's browser. Line 30 attempts to obtain the username/password from the user. Line 31 is executed if the users have already submitted their username/password combination and there was no corresponding match in the password file. |
| 32 | Print a message to the screen notifying the users that their username/password combination is invalid. |
| 33 | Exit from the script. |
| 34 | End the function declaration. |
| 35 | Begin the main program. |
| 36 | Include a comment in the script as to the nature of the <code>\$mode</code> variable. |
| 37 | Assign the <code>\$mode</code> variable the value "md5" (or "crypt" if your passwords are encrypted using the <code>crypt()</code> function). |
| 38 | Assign the location of the password file to the <code>\$password_file</code> variable. |
| 39 | Check to see if the global variable <code>\$PHP_AUTH_USER</code> has been set. <code>\$PHP_AUTH_USER</code> is a special PHP variable that is used with HTTP Authentication. This is the variable that is obtained when the user logs in using the HTTP Authentication window. |
| 40 | If <code>\$PHP_AUTH_USER</code> variable has not been set, then execute the <code>authenticate()</code> function. |
| 41–42 | If the <code>\$PHP_AUTH_USER</code> variable has been set, then execute the <code>check_pass()</code> function using the <code>\$PHP_AUTH_USER</code> , <code>\$PHP_AUTH_PW</code> , and <code>\$mode</code> variables as arguments. The <code>\$PHP_AUTH_PW</code> variable is the special PHP global variable obtained from the HTTP Authentication login window. |

Script 7–4 http_authentication.php Line-by-Line Explanation (Continued)

| LINE | DESCRIPTION |
|-------|--|
| 43–45 | If the <code>check_pass()</code> function returns true (1), then print out “ACCEPTED” to the screen. You would normally provide your protected content here. |
| 46–48 | If the <code>check_pass()</code> function returns false, execute the <code>authenticate()</code> function again. The users have three chances to correctly enter their username and password. After the third failure, the script returns lines 31 and 32. |
| 49 | Close the if statement started on line 39. |
| 50 | End the script. |

Authenticating Users by Database Query

The most common method of authentication for database-backed sites is to use the database. Why bother with clunky text files when you have the speed and ease of an SQL database at your fingertips?

Database-based authentication can use the same features as file-based authentication, such as `md5()` or `crypt()` encryption. Usernames and passwords are stored in a table on the database. You can store other information in this table as well, such as email addresses or first and last names, as you saw in the example applications in Chapter 5.

This next script provides a bare-bones approach to using a database to authenticate users. It uses plain-text passwords, but you can easily include encrypted passwords using the techniques from the earlier scripts in this chapter.

SCRIPT 7–5 DB_authenticate.php

```
1. <?
2.
3. /* SQL REQUIRED FOR THIS SCRIPT *****
4. create table users (
5.     id INT NOT NULL,
6.     username VARCHAR(16),
7.     password VARCHAR(8),
8.     primary key(id));
9. *****/
10.
11. function connect() {
```

SCRIPT 7-5 DB_authenticate.php (Continued)

```
12.     if(!$db =
13.         @mssql_pconnect("localhost","mssqluser","password")){
14.             print("<h1>Cannot Connect to the DB!</h1>\n");
15.         } else {
16.             mssql_select_db("php", $db);
17.             return 1;
18.         }
19.     }
20.
21.     function check_user($user, $password) {
22.         if(connect()) {
23.             $password = substr($password, 0, 8);
24.             $sql = "select * from users where username = '$user'
and password = '$password'";
25.             $result = mssql_query($sql);
26.             if (mssql_num_rows($result) == 1) {
27.                 setcookie("user",$user);
28.                 setcookie("password",$password);
29.                 return 1;
30.             } else {
31.                 ?>
32.                 <h3>Sorry, you are not authorized!</h3>
33.                 <?
34.                 return 0;
35.             }
36.         }
37.     }
38.
39.     /***** MAIN *****/
40.     if(!isset($user) or !check_user($user, $password)) {
41.         ?>
42.         <h1>You must log in to view this page</h1>
43.         <form action = "DB_authenticate.php" method="post">
44.         <P>Username: <input type="text" name="user"><br>
45.         Password: <input type="password" name="password"
maxlength="8" size="8"><br>
46.         <input type="submit" name="submit" value="Submit">
47.         </form>
48.         <?
49.     } else {
50.         ?>
51.         <h1>Authorized!</h1>
```

SCRIPT 7-5 DB_authenticate.php (Continued)

```
52.     <?  
53.     }  
54.     ?>
```

Script 7-5 DB_authenticate.php Line-by-Line Explanation

| LINE | DESCRIPTION |
|------|--|
| 3-9 | These lines provide the SQL statement to use to create a database table that works with this script. Once you have created your table, you can add users using standard SQL statements such as: <pre>insert into users values (1, 'username', 'password');</pre> Keep in mind that you need a unique ID for each user that you add. |
| 11 | Create a function called connect(). This function is used to create a connection to the database. |
| 12 | Attempt to connect to the database, in this case an MS SQL database. (You can replace all the MS SQL functions with MySQL functions simply by changing the first "s" in mssql to a "y"—"mysql".) |
| 13 | If the script cannot connect to the database, then print an error. |
| 14 | Return false (0) to the calling program, because the database connection failed. |
| 16 | If the connection attempt was successful, then select the database on the database server that the script will use. |
| 17 | Return true (1) to the calling program, because the connection was successful. |
| 18 | Close the if statement started on line 12. |
| 19 | End the function declaration. |
| 21 | Create a function caller check_user() that checks the user-entered username and password against a valid username and password that is stored in a database. The script takes as arguments \$user and \$password, both entered by a user through a login form. |
| 22 | Attempt to connect to the database using the connect function. If the connection succeeds, then continue to line 23. Otherwise, go to line 36, which is the end of the function. |
| 23 | Modify the user-entered password so that it matches the format used by the database. In this case, the script just truncates the password to eight characters. You may want to use one of the md5() or crypt() functions here if you stored encrypted passwords in your database. |

Script 7-5 DB_authenticate.php Line-by-Line Explanation (Continued)

| LINE | DESCRIPTION |
|-------|---|
| 24 | Generate an SQL statement that attempts to select a row from the “users” table that matches the user-entered username and password. |
| 25 | Run the SQL query and assign the result to the <code>\$result</code> variable. |
| 26 | Check to see if the result from the query has exactly one row, since usernames are supposed to be unique. (You didn’t put duplicate usernames in the database, did you?) |
| 27–28 | If the result did have exactly one row, meaning that the user-entered username and password match the same username and password in the database, then set two session cookies on users’ browsers so that they remain “logged in” the entire time they are using your protected pages. |
| 29 | Return true (1), since the authentication was successful. |
| 30–35 | If the result did not contain exactly one row, then the user-entered username and password did not match any in the database. Print an error message and return false to the calling program. |
| 36 | Close the if statement started on line 22. |
| 37 | End the function declaration. |
| 39 | Begin the main program. |
| 40–48 | Check to see if the <code>\$user</code> variable is set or if the user-entered username and password match any in the database via the <code>check_pass()</code> function. If the <code>\$user</code> variable has not been set, then we know that the current user has not attempted to log in yet. If the <code>\$user</code> variable is set, but the <code>check_user()</code> function returns false, then we know that the user entered an invalid password. In either case, print the login form so that the user may enter a correct username and password. |
| 49–53 | If the script gets to this point, it is because the <code>\$user</code> variable has been set and <code>check_user()</code> returned a true value. This must be a valid user! Display a note telling users that they are authorized. At this point, you would display your protected content. |