Chapter 17

Authenticating Users with a Database

Chapter 9 showed you the principles of user authentication and sessions to password protect parts of your website, but the login scripts all relied on usernames and passwords stored in text files. Keeping user details in a database is both more secure and more efficient. Instead of just storing a list of usernames and passwords, a database can store other details, such as first name, family name, email address, and so on. MySQL also gives you the option of using either one- or two-way encryption. In the first section of this chapter, we'll examine the difference between the two. Then you'll create registration and login scripts for both types of encryption.

What this chapter contains:

- Deciding how to encrypt passwords
- Using one-way encryption for user registration and login
- Using two-way encryption for user registration and login
- Decrypting passwords

Choosing an encryption method

The PHP solutions in Chapter 9 use the SHA-1 encryption algorithm. It offers a high level of security, particularly if used in conjunction with a **salt** (a random value that’s added to make decryption harder). SHA-1 is a one-way encryption method: once a password has been encrypted, there’s no way of converting it back to plain text. This is both an advantage and a disadvantage. It offers the user greater security because passwords encrypted this way remain secret. However, there’s no way of reissuing a lost password, since not even the site administrator can decrypt it. The only solution is to issue the user a temporary new password, and ask the user to reset it.

The alternative is to use two-way encryption, which relies on a pair of functions: one to encrypt the password and another to convert it back to plain text, making it easy to reissue passwords to forgetful users. Two-way encryption uses a secret key that is passed to both functions to perform the conversion. The key is simply a string that you make up yourself. Obviously, to keep the data secure, the key needs to be sufficiently difficult to guess and should never be stored in the database. However, you need to embed
the key in your registration and login scripts—either directly or through an include file—so if your scripts are ever exposed, your security is blown wide apart. MySQL offers a number of two-way encryption functions, but \texttt{AES\_ENCRYPT()} is considered the most secure. It uses the Advanced Encryption Standard with a 128-bit key length (AES-128) approved by the U.S. government for the protection of classified material up to the SECRET level (TOP SECRET material requires AES-192 or AES-256).

Both one-way and two-way encryption have advantages and disadvantages. Many security experts recommend that passwords should be changed frequently. So, forcing a user to change a forgotten password because it can’t be decrypted could be regarded as a good security measure. On the other hand, users are likely to be frustrated by the need to deal with a new password each time they forget the existing one. I’ll leave it to you to decide which approach is best suited to your circumstances, and I’ll concentrate solely on the technical implementation.

### Using one-way encryption

In the interests of keeping things simple, I’m going to use the same basic forms as in Chapter 9, so only the username, salt, and encrypted password are stored in the database.

### Creating a table to store users’ details

In phpMyAdmin, create a new table called \texttt{users} in the \texttt{phpsols} database. The table needs four columns (fields) with the settings listed in Table 17-1.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Length/Values</th>
<th>Attributes</th>
<th>Null</th>
<th>Index</th>
<th>A_I</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>INT</td>
<td></td>
<td>UNSIGNED</td>
<td>Deselected</td>
<td>PRIMARY</td>
<td>Selected</td>
</tr>
<tr>
<td>username</td>
<td>VARCHAR</td>
<td>15</td>
<td></td>
<td>Deselected</td>
<td>UNIQUE</td>
<td></td>
</tr>
<tr>
<td>salt</td>
<td>INT</td>
<td></td>
<td>UNSIGNED</td>
<td>Deselected</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pwd</td>
<td>CHAR</td>
<td>40</td>
<td></td>
<td>Deselected</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To ensure no one can register the same username as one that’s already in use, the \texttt{username} column is given an \texttt{UNIQUE} index.

In Chapter 9, the username doubled as the salt, but storing the details in a database means that you can choose something more unique and difficult to guess. Although a Unix timestamp follows a predictable pattern, it changes every second. So even if an attacker knows the day on which a user registered, there are 86,400 possible values for the salt, which would need to be combined with every attempt to guess the password. So the \texttt{salt} column needs to store an integer (INT).

The \texttt{pwd} column, which is where the encrypted password is stored, needs to be 40 characters long because the SHA-1 algorithm always produces an alphanumeric string of that length. It’s a fixed length, so \texttt{CHAR} is used in preference to \texttt{VARCHAR}. The \texttt{CHAR} data type is more efficient when dealing with fixed-length strings.