PHP Sessions Tutorial: Authentication with Password Hashing

This tutorial demonstrates how to use PHP sessions for authentication, including password hashing practices.

What are PHP Sessions?

Sessions allow you to store user-specific information on the server for later use across multiple pages. Unlike cookies, which store data on the client-side, sessions store data on the server, making them more secure for sensitive information.

Key Concepts of PHP Sessions

1. Starting a Session

To use sessions, you must first start a session using the `session_start()` function. This should be called at the beginning of your PHP script, before any output is sent to the browser.

```php
<?php
session_start();
?>
```

2. Setting Session Variables

After starting a session, you can set session variables using the `$_SESSION` superglobal array:

```php
<?php
$_SESSION['username'] = 'JohnDoe';
$_SESSION['user_id'] = 123;
?>
```

3. Retrieving Session Data

To access session data on other pages, you first call `session_start()`, then access the `$_SESSION` array:

```php
<?php
session_start();
$username = $_SESSION['username'];
echo "Welcome, " . $username;
?>
```

4. Destroying a Session

To end a session and remove all session data:

```php
<?php
session_start();
session_destroy();
?>
```

Practical Example: Authentication System

Let's create an authentication system using PHP sessions and password hashing. This system will include user registration, login, a protected page, and logout functionality.

1. User Registration (register.php)

First, let's create a registration page where users can create an account:

```php
<?php
session_start();

// Simple in-memory storage (replace with database in real-world scenario)
$users = [];

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    global users;

    $username = $_POST['username'];
    $password = $_POST['password'];

    // Hash the password
    $hashed_password = password_hash($password, PASSWORD_DEFAULT);

    // In a real application, you'd save this to a database
    $users[$username] = $hashed_password;

    $_SESSION['registration_success'] = "Registration successful. You can now log in.";
    header("Location: login.php");
    exit();
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register</title>
</head>
<body>
```

```
    <h2>Register</h2>
    <form action="register.php" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>

        <input type="submit" value="Register">
    </form>
    <p>Already have an account? <a href="login.php">Login here</a></p>
</body>
</html>
```

2. Login Form (login.php)

```
<?php
session_start();

// If user is already logged in, redirect to the protected page
if (isset($_SESSION['user_id'])) {
    header("Location: protected_page.php");
    exit();
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
</head>
<body>
    <h2>Login</h2>
    <?php
    if (isset($_SESSION['login_error'])) {
        echo "<p style='color: red;'>" . $_SESSION['login_error'] . "</p>";
        unset($_SESSION['login_error']);
    }
    if (isset($_SESSION['registration_success'])) {
        echo "<p style='color: green;'>" . $_SESSION['registration_success'] . "</p>";
        unset($_SESSION['registration_success']);
    }
    ?>
    <form action="login_process.php" method="post">
```

```
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>

        <input type="submit" value="Login">
    </form>
    <p>Don't have an account? <a href="register.php">Register here</a></p>
</body>
</html>
```

3. Login Processing Script (login_process.php)

```
<?php
session_start();

// In a real-world scenario, you would fetch user data from a database
// For this example, we'll use the in-memory storage from the registration process
global $users;

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST['username'];
    $password = $_POST['password'];

    if (isset($users[$username])) {
        // Verify the password
        if (password_verify($password, $users[$username])) {
            // Authentication successful
            $_SESSION['user_id'] = $username; // In reality, this would be a unique user ID
            $_SESSION['username'] = $username;
            header("Location: protected_page.php");
            exit();
        }
    }

    // Authentication failed
    $_SESSION['login_error'] = "Invalid username or password";
    header("Location: login.php");
    exit();
} else {
    header("Location: login.php");
    exit();
}
?>
```

4. Protected Page (protected_page.php)

```php
<?php
session_start();

// Check if user is logged in
if (!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Protected Page</title>
</head>
<body>
    <h2>Welcome to the Protected Page</h2>
    <p>Hello, <?php echo htmlspecialchars($_SESSION['username']); ?>!</p>
    <p>This is a protected page that only logged-in users can access.</p>
    <a href="logout.php">Logout</a>
</body>
</html>
```

5. Logout Script (logout.php)

```php
<?php
session_start();

// Unset all session variables
$_SESSION = array();

// Destroy the session
session_destroy();

// Redirect to the login page
header("Location: login.php");
exit();
?>
```

How This Authentication System Works

1. **User Registration (register.php):**
    - Allows users to create a new account.

- Hashes the password using `password_hash()` before storing it.
- In a real application, you would store this in a database instead of in-memory.

2. **Login Form (login.php):**
   - Displays the login form and any messages (errors or successful registration).
   - Provides a link to the registration page for new users.
3. **Login Processing (login_process.php):**
   - Retrieves the stored hashed password for the given username.
   - Uses `password_verify()` to check if the entered password matches the stored hash.
   - If authentication is successful, sets session variables and redirects to the protected page.
   - If authentication fails, sets an error message and redirects back to the login form.
4. **Protected Page (protected_page.php):**
   - Checks if the user is logged in by looking for the user_id session variable.
   - If not logged in, redirects to the login page.
   - If logged in, displays a welcome message and a logout link.
5. **Logout (logout.php):**
   - Destroys the session, effectively logging out the user.
   - Redirects back to the login page.

Security Considerations

- **Password Hashing:** We use `password_hash()` to securely hash passwords before storage. This function automatically handles salting and uses a strong algorithm (currently bcrypt).
- **Password Verification:** We use `password_verify()` to check passwords against their hashed versions, which is safe against timing attacks.
- **HTTPS:** Always use HTTPS in a production environment to encrypt data transmitted between the client and server.
- **Session Management:** Consider using session regeneration (`session_regenerate_id()`) after login to prevent session fixation attacks.
- **CSRF Protection:** Implement CSRF (Cross-Site Request Forgery) protection for forms in a real-world application.
- **Input Validation:** Always validate and sanitize user inputs to prevent SQL injection and other attacks.
- **Error Messages:** Be cautious about the information revealed in error messages to prevent username enumeration.

Conclusion

This tutorial demonstrates how to create an authentication system using PHP sessions and proper password hashing techniques. While this example uses in-memory storage for simplicity, a real-world application would use a database to store user information securely. The principles of session management and password hashing shown here form the foundation of user authentication in PHP applications.