CAPÍTULO 12

Cookies, sesiones y autenticación

A medida que tus proyectos web se hagan más grandes y complicados, tendrás la creciente necesidad de hacer un seguimiento de tus usuarios. Incluso si no ofreces inicios de sesión y contraseñas, aún tendrás que almacenar con frecuencia detalles sobre la sesión en curso de un usuario y posiblemente también reconocerlo cuando regrese a tu sitio.

Hay varias tecnologías que soportan estos tipos de interacciones, que van desde las simples cookies del navegador a la gestión de sesiones y autenticación HTTP. Entre todas ellas, te ofrecen la oportunidad de que configures tu sitio según las preferencias de tus usuarios y aseguran una transición suave y agradable del mismo.

Uso de cookies en PHP

Una *cookie* es un elemento de datos que un servidor web guarda en el disco duro de tu ordenador a través de un navegador web. Puede contener casi cualquier información alfanumérica (siempre y cuando no alcance los 4 KB) y puede recuperarse de tu ordenador y ser devuelto al servidor. Entre los usos más habituales se incluyen el seguimiento de sesiones, el mantenimiento de datos de visitas, la conservación del contenido de la cesta de la compra, el almacenamiento de los datos de inicio de sesión, etc.

Debido a sus implicaciones en lo que se refiere a la privacidad, las cookies solo pueden leerse desde el sitio web de la empresa que las ha emitido. En otras palabras, si una cookie la emite, por ejemplo, oreilly.com, o marcombo.com la puede recuperar solo un servidor web que utilice ese dominio. Esto evita que otros sitios web accedan a datos para los que no están autorizados.

Debido a la forma en que funciona Internet, se pueden incrustar varios elementos en una página web desde múltiples dominios, cada uno de los cuales puede emitir sus propias cookies. Cuando esto sucede, se denominan cookies de terceros. Más comúnmente, estas las crean empresas de publicidad con el fin de rastrear a los usuarios a través de múltiples sitios web.

Debido a esta posibilidad, la mayoría de los navegadores permiten a los usuarios desactivar las cookies tanto para el dominio del servidor actual, servidores de terceros, o ambos. Afortunadamente, la mayoría de las personas que inhabilitan las cookies lo hacen solo para sitios web de terceros.

Las cookies se intercambian durante la transferencia de encabezamientos, antes de que se envíe el HTML actual de una página web, y es imposible enviar una cookie una vez que se ha enviado cualquier código HTML. Por lo tanto, es importante planificar cuidadosamente el uso de cookies. La Figura 12-1 ilustra un cuadro de diálogo típico de solicitud y respuesta entre un navegador web y un servidor web que pasa las cookies.

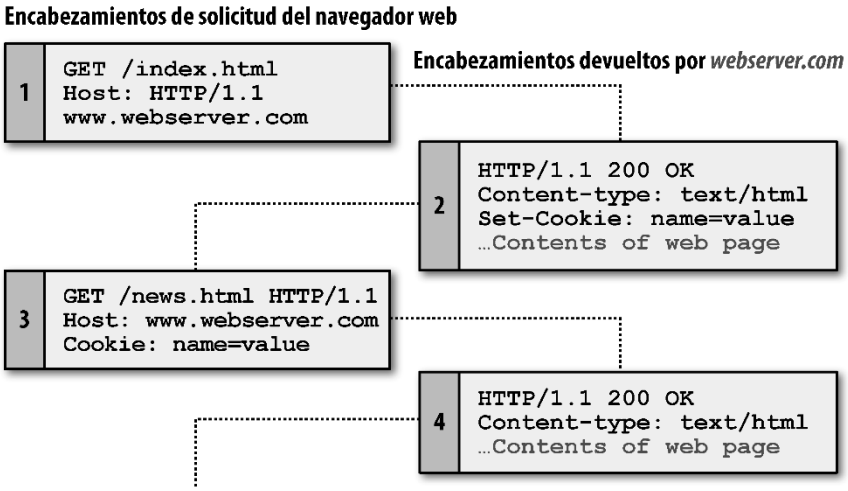

Figura 12-1. Diálogo con cookies pregunta/respuesta entre navegador/servidor

Este intercambio muestra un navegador que recibe dos páginas:

1. El navegador emite una solicitud para recoger la página principal, *index.html*, del sitio web *http://www.webserver.com*. El primer encabezamiento especifica el archivo, y el segundo especifica el servidor.
2. Cuando el servidor web en *webserver.com* recibe este par de encabezamientos, devuelve algunos de los suyos. El segundo encabezamiento define el tipo de contenido a enviar (`text/html`), y el tercero envía una cookie con el nombre `name` y con el valor `value`. Solo entonces se transfieren los contenidos de la página web.
3. Una vez que el navegador ha recibido la cookie, la devolverá con cada solicitud futura realizada al servidor emisor hasta que la cookie expire o sea eliminada. Así que, cuando el navegador solicita la nueva página */news.html*, también devuelve la cookie `name` con el valor `value`.
4. Debido a que la cookie ya se ha configurado, cuando el servidor recibe la solicitud de envío */news.html*, no tiene que reenviar la cookie, sino que simplemente devuelve la página solicitada.

Es fácil editar las cookies directamente desde el navegador utilizando herramientas integradas para desarrolladores, o mediante extensiones. Por lo tanto, debido a que los usuarios pueden cambiar los valores de las cookies, no debes poner información clave como son nombres de usuario en una cookie, o te enfrentas a la posibilidad de que sean manipuladas en formas que no esperas. El mejor uso de las cookies es para almacenar datos como el idioma o la configuración de la moneda.

Configuración de cookies

Configurar una cookie en PHP es sencillo. Siempre y cuando no se haya transferido ningún HTML, puedes llamar a la función `setcookie`, que tiene la siguiente sintaxis (ver Tabla 12-1):

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

Tabla 12-1. Parámetros de setcookie

Parámetro	Descripción	Ejemplo
`name`	El nombre de la cookie. Este es el nombre que utilizará tu servidor para acceder a la cookie en posteriores solicitudes del navegador.	`location`
`value`	El valor de la cookie, o los contenidos de la cookie. Este parámetro puede contener hasta 4 KB de texto alfanumérico.	`USA`
`expire`	(*Opcional*). La indicación Unix de fecha y hora de la fecha de caducidad. En general, probablemente se utilizará `time()` más una cantidad de segundos. Si no se indica, la cookie expira cuando el navegador se cierra.	`time() + 2592000`
`path`	(*Opcional*). La ruta de la cookie en el servidor. Si se trata de una `/` (barra diagonal), la cookie está disponible en todo el dominio como *www.webserver.com*. Si es un subdirectorio, la cookie solo está disponible dentro de ese subdirectorio. El valor predeterminado es el directorio actual en el que está configurada la cookie, esta es la configuración que normalmente usará.	`/`
`domain`	(*Opcional*). El dominio Internet de la cookie. Si es *webserver.com*, la cookie estáadisponible para todo *webserver.com* y sus subdominios, como *www.webserver.com* e *images.webserver.com*. Si es *images.webserver.com*, la cookie está disponible solo para *images.webserver.com* y sus subdominios como *sub.images.webserver.com*, pero no, por ejemplo, para *www.webserver.com*.	`webserver.com`
`secure`	(*Opcional*). Si la `cookie` debe usar una conexión segura (*https://*). Si este valor es `TRUE`, la cookie se puede transferir solo a través de una conexión segura. El valor predeterminado es `FALSE`.	`FALSE`
`httponly`	(*Opcional*). Implementado a partir de la versión 5.2.0. de PHP). Si la cookie debe usar el protocolo HTTP. Si este valor es `TRUE`, los lenguajes de script como JavaScript no pueden acceder a la cookie(no es compatible con todos los navegadores). El valor predeterminado es `FALSE`.	`FALSE`

Por lo tanto, para crear una cookie con el nombre `location` y el valor `USA` que sea accesible a través de todo el servidor web en el dominio actual, y que se eliminará de la memoria caché del navegador en siete días, utiliza lo siguiente:

```
setcookie('location', 'USA', time() + 60 * 60 * 24 * 7, '/');
```

Acceso a cookies

Leer el valor de una cookie es tan sencillo como acceder a la matriz del sistema `$_COOKIE`. Por ejemplo, si deseas ver si el navegador actual tiene la cookie llamada `location` ya almacenada y, en caso afirmativo, leer su valor, utiliza lo siguiente:

```
if (isset($_COOKIE['location'])) $location = $_COOKIE['location'];
```

Ten en cuenta que puedes volver a leer una cookie solo después de que se haya enviado a un navegador web. Esto significa que cuando emites una cookie, no puedes volver a leerla hasta que el navegador recarga la página (u otra con acceso a la cookie) desde tu sitio web y en el proceso pasa la cookie de nuevo al servidor.

Eliminación de cookies

Para eliminar una cookie, debes volver a emitirla y fijar una fecha pasada. Es importante que todos los parámetros en tu nueva llamada a `setcookie`, excepto la indicación de fecha y hora, sean idénticos a los parámetros de la cookie cuando se emitió por primera vez, ya que de lo contrario, no se producirá su eliminación. Por lo tanto, para eliminar la cookie creada anteriormente, usarías lo siguiente:

```
setcookie('location', 'USA', time() - 2592000, '/');
```

Mientras el tiempo que se informa sea un tiempo pasado, la cookie debe borrarse. Sin embargo, he utilizado un tiempo de 2 592 000 segundos (un mes) pasado en caso de que el ordenador del cliente la fecha y la hora no estén ajustadas correctamente. También puedes proporcionar una cadena vacía para la cookie (o el valor `FALSE`), y PHP establecerá automáticamente su tiempo en el pasado.

Autenticación HTTP

La autenticación HTTP utiliza el servidor web para la gestión de los usuarios y las contraseñas de la aplicación. Es adecuada para aplicaciones sencillas que piden a los usuarios que inicien sesión, aunque la mayoría de las aplicaciones tendrán necesidades especiales o requisitos de seguridad más estrictos que requieren otras técnicas.

Para usar la autenticación HTTP, PHP envía una petición de encabezamiento solicitando que se inicie un diálogo de autenticación con el navegador. El servidor debe tener esta característica activada para que funcione, pero debido a que es tan habitual, es probable que tu servidor ofrezca esa característica.

Aunque normalmente se instala con Apache, es posible que el módulo de autenticación HTTP no se instale necesariamente en el servidor que utilizas. Por lo tanto, intentar ejecutar estos ejemplos puede generar un error que te indique que la función no está habilitada, en cuyo caso debes instalar el módulo y cambiar el archivo de configuración para cargarlo, o solicitar al administrador del sistema que realice estas correcciones.

Después de que el usuario introduzca tu URL en el navegador o visite la página a través de un enlace, verá un mensaje emergente, "Authentication Required" (Autenticación requerida), y solicitará dos campos: nombre de usuario y contraseña (la Figura 12-2 ilustra cómo se presenta en Firefox).

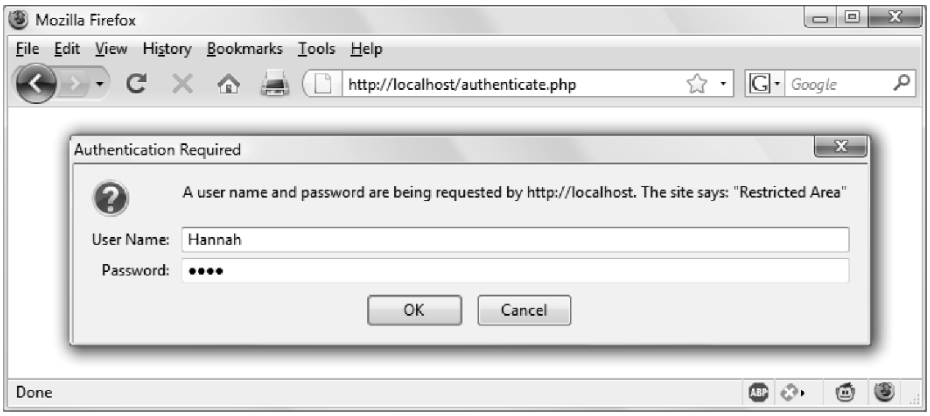

Figura 12-2. Solicitud de inicio de sesión de autenticación HTTP

El Ejemplo 12-1 muestra el código correspondiente.

Ejemplo 12-1. Autenticación PHP

```
<?php
  if (isset($_SERVER['PHP_AUTH_USER']) &&
      isset($_SERVER['PHP_AUTH_PW']))
  {
    echo "Welcome User: " . htmlspecialchars($_SERVER['PHP_AUTH_USER']) .
         " Password: "    . htmlspecialchars($_SERVER['PHP_AUTH_PW']);
  }
  else
  {
    header('WWW-Authenticate: Basic realm="Restricted Area"');
    header('HTTP/1.0 401 Unauthorized');
    die("Please enter your username and password");
  }
?>
```

Lo primero que hace el programa es buscar dos valores de la matriz en particular: `$_SERVER['PHP_AUTH_USER']` y `$_SERVER['PHP_AUTH_PW']`. Si ambos existen, representan el nombre de usuario y la contraseña introducidos por un usuario en una solicitud de autenticación.

Observa que cuando se visualizan en la pantalla, los valores que se han devuelto en la matriz `$_SERVER` se procesan primero a través de la función `htmlspecialchars`. Esto se debe a que estos valores los ha introducido el usuario y, por lo tanto, no pueden ser de confianza, ya que un hacker podría intentar una secuencia de scripting entre sitios añadiendo caracteres HTML y otros símbolos a la entrada. `htmlspecialchars` traduce cualquier entrada de este tipo en entidades HTML inofensivas.

Si ninguno de los dos valores existe, el usuario todavía no se ha autenticado y tú puedes ver el aviso, como el de la Figura 12-2, que emite el encabezamiento siguiente, en el que `Basic realm` es el nombre de la sección que está protegida y que aparece como parte del aviso emergente:

```
WWW-Authenticate: Basic realm="Restricted Area"
```

Si el usuario rellena los campos, el programa PHP se ejecuta de nuevo desde el principio. Pero si el usuario hace clic en el botón Cancel, el programa pasa a las dos líneas siguientes y envía el siguiente encabezado y un mensaje de error:

```
HTTP/1.0 401 Unauthorized
```

La declaración `die` hace que se visualice el texto "Please, enter your username and password" (ver la Figura 12-3).

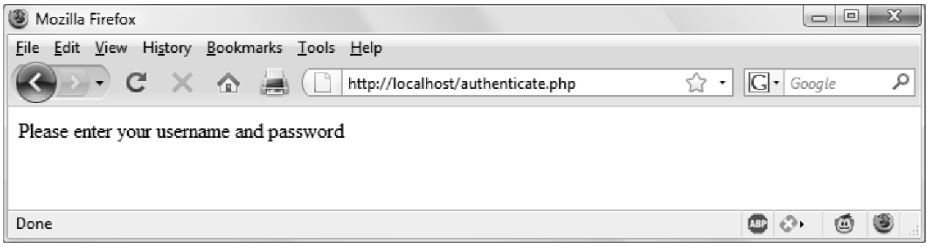

Figura 12-3. Resultado de hacer clic en el botón Cancel

Una vez que un usuario se ha autenticado, no podrás conseguir que el cuadro de diálogo de autenticación aparezca de nuevo, a menos que el usuario cierre y reabra todas las ventanas del navegador, porque el navegador web se mantendrá devolviendo el mismo nombre de usuario y contraseña a PHP. Es posible que necesites cerrar y volver a abrir tu navegador unas cuantas veces mientras trabajas en esta sección y probar diferentes cosas. La manera más fácil de hacerlo es abrir una nueva ventana privada o anónima para ejecutar estos ejemplos, por lo que no tendrás que cerrar el navegador.

Ahora vamos a comprobar si hay un nombre de usuario y contraseña válidos. No se requiere que cambies mucho el código del Ejemplo 12-1 para añadir esta comprobación, además de modificar código de mensaje de bienvenida anterior para comprobar si el nombre de usuario y la contraseña son correctos y, a continuación, emitir un mensaje de bienvenida. Una autenticación errónea provoca el envío de un mensaje de error (ver Ejemplo 12-2).

Ejemplo 12-2. Autenticación PHP con control de entrada

```
<?php
  $username = 'admin';
  $password = 'letmein';

  if (isset($_SERVER['PHP_AUTH_USER']) &&
      isset($_SERVER['PHP_AUTH_PW']))
  {
    if ($_SERVER['PHP_AUTH_USER'] === $username &&
        $_SERVER['PHP_AUTH_PW']   === $password)
          echo "You are now logged in";
    else die("Invalid username/password combination");
  }
  else
  {
    header('WWW-Authenticate: Basic realm="Restricted Area"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Please enter your username and password");
  }
?>
```

Cuando se comparan nombres de usuario y contraseñas se utiliza el operador `===` (identidad), en lugar del operador `==` (igualdad). Esto se debe a que estamos comprobando si los dos los valores coinciden *exactamente*. Por ejemplo, `'0e123' == '0e456'`, y esta comparación no es adecuada para comprobar nombres de usuario o contraseñas.

Ahora existe un mecanismo para autenticar a los usuarios, pero solo para un único nombre de usuario y contraseña. Además, la contraseña aparece en texto sin cifrar dentro del archivo PHP, y si alguien consigue hackear tu servidor, lo sabría al instante. Así que, vamos a buscar una mejor manera de gestionar los nombres de usuario y las contraseñas.

Almacenamiento de nombres de usuario y contraseñas

Obviamente, MySQL es la forma natural de almacenar nombres de usuario y contraseñas. Pero, lo repito, no queremos almacenar las contraseñas con texto sin cifrar, porque nuestro sitio web podría estar comprometido si un hacker accediera a la base de datos. En lugar de esto, usaremos un buen truco llamado *función unidireccional*.

Este tipo de función es fácil de usar y convierte una cadena de texto en un texto aparentemente aleatorio. Debido a su naturaleza unidireccional, tales funciones son imposibles de invertir (revertir), por lo que su salida puede almacenarse de forma segura en una base de datos, y cualquiera que la sustraiga no podrá saber más en lo que se refiere a las contraseñas que se utilizan.

En ediciones anteriores de este libro, recomendaba el uso del algoritmo de hash MD5 para la seguridad de los datos. El tiempo pasa, sin embargo, y ahora MD5 se considera fácilmente pirateable y, por lo tanto, es inseguro. De hecho, incluso su sustituto, recomendado anteriormente *SHA-1* se puede, aparentemente, hackear.

Así que, ahora que PHP 5.5 es más o menos el estándar mínimo en todas partes, he pasado a utilizar la función de hash integrada, que es mucho más segura y gestiona todo automáticamente de una manera ordenada.

Anteriormente, para almacenar una contraseña de forma segura, habrías tenido que añadirle *sal*, que es un término que se refiere a incorporar caracteres adicionales a una contraseña, y que el usuario no ha introducido (para hacerla aún más impenetrable). Entonces necesitarías ejecutar el resultado a través de la función unidireccional para convertirla en un conjunto de caracteres aleatorios, que sería difícil de descifrar.

Por ejemplo, un código como el siguiente (que ahora es muy inseguro debido a la velocidad y potencia que tienen las modernas unidades de procesamiento gráfico):

```
echo hash('ripemd128', 'saltstringmypassword');
```

mostraría este valor:

```
9eb8eb0584f82e5d505489e6928741e7
```

Recuerda que este no es un método recomendado que debes utilizar siempre. Trátalo como un ejemplo de lo que no hay que hacer, ya que es muy inseguro. En lugar de eso, sigue leyendo.

Uso de password_hash

A partir de la versión 5.5 de PHP, hay una forma mucho más recomendable de crear funciones hash de contraseñas saladas: la función `password_hash`. Proporciona `PASSWORD_DEFAULT` como su segundo argumento (obligatorio) para pedir a la función que seleccione la función de hash más segura disponible en ese momento. `password_hash` también elegirá una sal al azar para cada contraseña. (No te sientas tentado de añadir más sal por tu cuenta, ya que esto podría comprometer el algoritmo de seguridad). Entonces, el siguiente código:

```
echo password_hash("mypassword", PASSWORD_DEFAULT);
```

devolverá una cadena como la siguiente, que incluye la sal y toda la información requerida para la verificación de la contraseña:

```
$2y$10$k0YljbC2dmmCq8WKGf8oteBGiXlM9Zx0ss4PEtb5kz22EoIkXBtbG
```

Si dejas que PHP escoja el algoritmo hash automáticamente, debes permitir que el hash devuelto se expanda en tamaño con el tiempo a medida que se implementa una mayor seguridad. Los desarrolladores de PHP recomiendan almacenar hashes en un campo de base de datos que puede expandirse como mínimo a 255 caracteres (aunque 60-72 es la longitud media correcta actualmente). Si lo deseas, puedes seleccionar manualmente el algoritmo BCRYPT para garantizar una cadena hash de solo 60 caracteres, suministrando la constante `PASSWORD_BCRYPT` como segundo argumento de la función. Sin embargo, no lo recomiendo a menos que tengas una razón muy poderosa.

Puedes proporcionar opciones (en forma de un tercer argumento optativo) para personalizar aún más la forma de calcular los hashes, como el coste o la cantidad de tiempo de procesador que hay que asignar al hashing (más tiempo significa más seguridad, pero un servidor más lento). El coste tiene un valor por defecto de `10`, que es el mínimo que deberías usar con BCRYPT.

Sin embargo, no quiero confundirte con más información de la que necesitas para ser capaz de almacenar los hashes de contraseñas de forma segura y con el mínimo de complicaciones, así que por favor, consulta la documentación (*http://php.net/password-hash*) si deseas más detalles sobre las opciones disponibles. Incluso puedes elegir tus propias sales (aunque se trata de un procedimiento obsoleto desde la versión de PHP 7.0 en adelante, ya que no se considera seguro, así que no te sientas tentado).

Uso de password_verify

Para verificar que una contraseña coincide con un hash, utiliza la función `password_verify`, pasa la cadena de contraseña que un usuario acaba de introducir y el valor hash almacenado para la contraseña de ese usuario (generalmente se recupera de tu base de datos).

Por lo tanto, supongamos que el usuario haya introducido previamente la contraseña (muy insegura) *mypassword*, ahora que tienes la cadena hash de su contraseña (de cuando el usuario la creó) almacenada en la variable `$hash`, puedes verificar que coinciden de la siguiente manera:

```
if (password_verify("mypassword", $hash)) echo "Valid";
```

Si se ha proporcionado la contraseña correcta para el hash, `password_verify` devuelve el valor `TRUE`, por lo que esta sentencia `if` mostrará la palabra "Valid". Si no coincide, entonces se devuelve `FALSE` y puedes pedir al usuario que lo intente de nuevo.

Programa de ejemplo

Veamos cómo actúan estas funciones juntas cuando se combinan con MySQL. Primero necesitas crear una nueva tabla para almacenar cada hash de las contraseñas, así que escribe el programa del Ejemplo 12-3 y guárdalo como *setupusers.php* (o descárgalo del sitio web complementario, *www.marcombo.info*) y luego ábrelo en tu navegador.

Ejemplo 12-3. Creación de una tabla de usuarios y adición de dos cuentas

```
<?php // setupusers.php
  require_once 'login.php';
  $connection = new mysqli($hn, $un, $pw, $db);

  if ($connection->connect_error) die("Fatal Error");

  $query = "CREATE TABLE users (
    forename VARCHAR(32) NOT NULL,
    surname  VARCHAR(32) NOT NULL,
    username VARCHAR(32) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
  )";

  $result = $connection->query($query);
  if (!$result) die("Fatal Error");

  $forename = 'Bill';
  $surname  = 'Smith';
  $username = 'bsmith';
  $password = 'mysecret';
  $hash     = password_hash($password, PASSWORD_DEFAULT);

  add_user($connection, $forename, $surname, $username, $hash);

  $forename = 'Pauline';
  $surname  = 'Jones';
  $username = 'pjones';
  $password = 'acrobat';
  $hash     = password_hash($password, PASSWORD_DEFAULT);

  add_user($connection, $forename, $surname, $username, $hash);

  function add_user($connection, $fn, $sn, $un, $pw)
  {
    $stmt = $connection->prepare('INSERT INTO users VALUES(?,?,?,?)');
    $stmt->bind_param('ssss', $fn, $sn, $un, $pw);
    $stmt->execute();
    $stmt->close();
  }
?>
```

Este programa creará la tabla *users* dentro de tu base de datos *publications* (o de cualquier base de datos que hayas configurado para el archivo *login.php* en el Capítulo 10). En esta tabla, el programa creará dos usuarios: Bill Smith y Pauline Jones. Tienen los nombres de usuario y contraseñas de *bsmith/mysecret* y *pjones/acrobat*, respectivamente.

Utilizando los datos de esta tabla, ahora podemos modificar el Ejemplo 12-2 para proceder a una correcta autentificación, y el Ejemplo 12-4 muestra el código necesario para conseguirlo. Introdúcelo o descárgalo desde el sitio web complementario (*www.marcombo.info*) y, a continuación, asegúrate de que está guardado como *authenticate.php* y llámalo en tu navegador.

Ejemplo 12-4. Autenticación PHP con MySQL

```
<?php // authenticate.php
  require_once 'login.php';
  $connection = new mysqli($hn, $un, $pw, $db);

  if ($connection->connect_error) die("Fatal Error");

  if (isset($_SERVER['PHP_AUTH_USER']) &&
      isset($_SERVER['PHP_AUTH_PW']))
  {
    $un_temp = mysql_entities_fix_string($connection,
       $_SERVER['PHP_AUTH_USER']);
    $pw_temp = mysql_entities_fix_string($connection,
       $_SERVER['PHP_AUTH_PW']);
    $query   = "SELECT * FROM users WHERE username='$un_temp'";
    $result  = $connection->query($query);

    if (!$result) die("User not found");
    elseif ($result->num_rows)
    {
      $row = $result->fetch_array(MYSQLI_NUM);

      $result->close();

      if (password_verify($pw_temp, $row[3])) echo
        htmlspecialchars("$row[0] $row[1] :
        Hi $row[0], you are now logged in as '$row[2]'");
      else die("Invalid username/password combination");
    }
    else die("Invalid username/password combination");
  }
  else
  {
    header('WWW-Authenticate: Basic realm="Restricted Area"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Please enter your username and password");
}
```

```
  $connection->close();

  function mysql_entities_fix_string($connection, $string)
  {
    return htmlentities(mysql_fix_string($connection, $string));
  }

  function mysql_fix_string($connection, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $connection->real_escape_string($string);
  }
?>
```

El uso de la autenticación HTTP impondrá aproximadamente 80 ms de penalización al usar `password_verify` con contraseñas hash con BCRYPT, con un coste por defecto de 10. Esta ralentización sirve como barrera para evitar que los atacantes intenten descifrar las contraseñas lo más rápidamente posible. Por lo tanto, la autenticación HTTP no es un buena solución en sitios muy concurridos, donde probablemente preferirás utilizar las sesiones (ver la siguiente sección).

Como es de esperar en este punto del libro, algunos de estos ejemplos están empezando a ser un poco más extensos. Pero no te desanimes. Las últimas 10 líneas son simplemente del Ejemplo 10-21 en el Capítulo 10. Están ahí para desinfectar la entrada de datos del usuario, lo cual es muy importante.

Las únicas líneas para preocuparse realmente en este punto son las que se destacan en negrita. Comienzan con la asignación de dos variables, `$un_temp` y `$pw_temp`, utilizando el nombre de usuario y la contraseña enviados. A continuación, se emite una consulta a MySQL para buscar el usuario `$un_temp` y, si se devuelve un resultado, asigna la primera fila a `$row`. Debido a que los nombres de usuario son únicos, solo habrá una fila.

Ahora todo lo que se necesita es comprobar el valor hash almacenado en la base de datos, que está en la cuarta columna (columna 3 cuando se empieza desde 0). `$row[3]` contiene el valor hash anterior calculado con `password_hash` cuando el usuario creó sus contraseña.

Si el hash y la contraseña que acaba de proporcionar el usuario coinciden, `password_verify` hará lo siguiente: devuelve `TRUE` y se emitirá una cadena de amable bienvenida, que llama al usuario por su nombre (ver Figura 12-4). De lo contrario, aparece un mensaje de error.

Puedes probarlo llamando al programa en tu navegador e introduciendo el nombre de usuario `bsmith` y la contraseña `mysecret` (o `pjones` y `acrobat`), los valores que se guardaron en la base de datos con el Ejemplo 12-3.

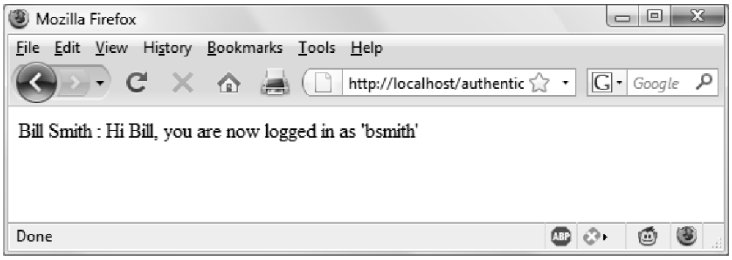

Figura 12-4. Bill Smith se ha autenticado

Al desinfectar la entrada inmediatamente después de que aparezca, podrás bloquear cualquier ataque malicioso de HTML, JavaScript o MySQL antes de que puedan llegar más lejos y no tendrás que desinfectar estos datos de nuevo. Si un usuario tiene caracteres como `<` o `&` en su contraseña (por ejemplo), estos se expandirán a `<` o `&` por la función `htmlentities`, pero siempre que tu código permita cadenas que pueden terminar siendo más grandes que el ancho de entrada proporcionado, y siempre y cuando ejecutes las contraseñas a través de esta desinfección, te sentirás más seguro.

Uso de sesiones

Debido a que tu programa no puede decir qué variables se establecieron en otros programas, o incluso qué valores estableció el mismo programa la última vez que se ejecutó, a veces querrás rastrear lo que sus usuarios están haciendo de una página web a otra. Puedes hacerlo si configuras campos ocultos en un formulario, como se ve en el Capítulo 10, y verificas los valores de los campos después de que se envíe el formulario, pero PHP proporciona una solución mucho más potente, más segura y más sencilla en forma de *sesiones*. Estas son grupos de variables que se almacenan en el servidor, pero se refieren solo al usuario actual. Para garantizar que las variables adecuadas se aplican a los usuarios correctos, PHP guarda una cookie en los navegadores web de los usuarios para identificarlos de manera única.

Esta cookie solo tiene significado para el servidor web y no se puede utilizar para averiguar ninguna información sobre el usuario. Puedes preguntar sobre los usuarios que tienen cookies desactivadas. Bueno, en estos tiempos, cualquier persona con cookies desactivadas no debería esperar tener la mejor experiencia de navegación; si las encuentras desactivadas, probablemente deberías informar a dicho usuario de que es necesario que las cookies estén activadas si desea beneficiarse plenamente de tu sitio, en lugar de tratar de encontrar formas de evitar el uso de cookies, que podrían crear problemas de seguridad.

Inicio de sesión

Iniciar una sesión requiere llamar a la función PHP `session_start` antes de que se haya generado cualquier HTML, de forma similar a como se envían las cookies durante los intercambios de encabezados.

Luego, para comenzar a guardar variables de sesión, solo tienes que asignarlas como parte de la matriz `$_SESSION`, así:

```
$_SESSION['variable'] = $value;
```

Después, se pueden volver a leer con la misma facilidad en posteriores ejecuciones de programa, como este caso:

```
$variable = $_SESSION['variable'];
```

Ahora supón que tienes una aplicación que siempre necesita acceso al nombre y apellido de cada usuario, tal y como se almacena en la tabla *users*, que deberías haber creado un poco antes. Modifiquemos más a fondo *authenticate.php* del Ejemplo 12-4 para configurar una sesión una vez que el usuario se ha autenticado.

El Ejemplo 12-5 muestra los cambios necesarios. La única diferencia es el contenido de la sección **`if (password_verify($pw_temp, $row[3]))`**, con la que empezamos abriendo una sesión y guardando estas variables en ella. Escribe este programa (o modifica el Ejemplo 12-4) y guárdalo como *authenticate2.php.* Pero no lo ejecutes en tu navegador todavía, ya que también tendrás que crear un segundo programa dentro de un momento.

Ejemplo 12-5. Configuración de una sesión después de una correcta autenticación

```
<?php // authenticate2.php
  require_once 'login.php';
  $connection = new mysqli($hn, $un, $pw, $db);

  if ($connection->connect_error) die("Fatal Error");

  if (isset($_SERVER['PHP_AUTH_USER']) &&
      isset($_SERVER['PHP_AUTH_PW']))
  {
    $un_temp = mysql_entities_fix_string($connection,
      $_SERVER['PHP_AUTH_USER']);
    $pw_temp = mysql_entities_fix_string($connection,
      $_SERVER['PHP_AUTH_PW']);
    $query   = "SELECT * FROM users WHERE username='$un_temp'";
    $result  = $connection->query($query);

    if (!$result) die("User not found");
    elseif ($result->num_rows)
    {
      $row = $result->fetch_array(MYSQLI_NUM);
```

```
      $result->close();

      if (password_verify($pw_temp, $row[3]))
      {
        session_start();
        $_SESSION['forename'] = $row[0];
        $_SESSION['surname']  = $row[1];
        echo htmlspecialchars("$row[0] $row[1] : Hi $row[0],
          you are now logged in as '$row[2]'");
        die ("<p><a href='continue.php'>Click here to continue</a></p>");
      }
      else die("Invalid username/password combination");
    }
    else die("Invalid username/password combination");
  }
  else
  {
    header('WWW-Authenticate: Basic realm="Restricted Area"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Please enter your username and password");
  }

  $connection->close();

  function mysql_entities_fix_string($connection, $string)
  {
    return htmlentities(mysql_fix_string($connection, $string));
  }

  function mysql_fix_string($connection, $string)
  {
    if (get_magic_quotes_gpc()) $string = stripslashes($string);
    return $connection->real_escape_string($string);
  }
?>
```

Otra adición al programa es el enlace "Click here to continue" ("Haga clic aquí para continuar") con el URL de destino *continue.php*. Este enlace se usará para ilustrar cómo la sesión se transferirá a otro programa o página web PHP. Por lo tanto, crea *continue.php* escribiendo el programa del Ejemplo 12-6 y guárdalo.

Ejemplo 12-6. Recuperación de las variables de sesión

```
<?php // continue.php session_start();

  if (isset($_SESSION['forename']))
  {
    $forename = htmlspecialchars($_SESSION['forename']);
    $surname  = htmlspecialchars($_SESSION['surname']);

    echo "Welcome back $forename.<br>
          Your full name is $forename $surname.<br>";
```

```
  }
  else echo "Please <a href=authenticate2.php>click here</a> to log in.";
?>
```

Ahora estás preparado para llamar a *authenticate2.php* y abrirlo en tu navegador. Introduce el nombre de usuario `bsmith` y la contraseña `mysecret` (o `pjones` y `acrobat`) cuando se te pida, y haz clic en el enlace para cargar *continue.php*. Cuando tu navegador lo llame, el resultado debería ser algo como la Figura 12-5.

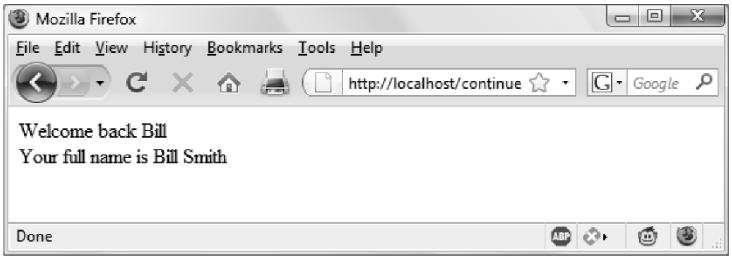

Figura 12-5. Mantenimiento de datos de usuario con sesiones

Las sesiones limitan prácticamente a un solo programa el extenso código necesario para que un usuario se autentique e inicie sesión. Una vez que se ha autenticado un usuario y has creado una sesión, el código de tu programa se convierte en algo muy sencillo. Solo tienes que llamar a `session_start` y buscar en `$_SESSION` cualquier variable a la que necesites acceder.

En el Ejemplo 12-6, una prueba rápida de si `$_SESSION['forename']` tiene un valor es suficiente para saber que el usuario actual está autenticado, porque las variables de sesión se almacenan en el servidor (a diferencia de las cookies, que se almacenan en el navegador web) y, por lo tanto, se puede confiar en ellas.

Si no se ha asignado un valor a `$_SESSION['forename']`, no hay ninguna sesión activa, por lo que la última línea de código del Ejemplo 12-6 dirige a los usuarios a la página de inicio de sesión en *authenticate2.php*.

Finalización de sesión

Cuando llega el momento de finalizar una sesión, normalmente cuando un usuario solicita cerrar la sesión de tu sitio, puedes utilizar la función `session_destroy`, como en el Ejemplo 12-7. Este ejemplo proporciona una función muy útil para destruir definitivamente una sesión, cerrar la sesión de usuario y desactivar todas las variables de sesión.

Ejemplo 12-7. Una función práctica para destruir una sesión y sus datos

```
<?php
  function destroy_session_and_data()
  {
    session_start();
```

```
    $_SESSION = array();
    setcookie(session_name(), '', time() - 2592000, '/');
    session_destroy();
  }
?>
```

Para poder ver cómo funciona, podrías modificar *continue.php* como en el Ejemplo 12-8.

Ejemplo 12-8. Recuperación de las variables de sesión y posterior destrucción de la sesión

```
<?php
  session_start();

  if (isset($_SESSION['username']))
  {
    $forename = $_SESSION['forename'];
    $surname  = $_SESSION['surname'];

    destroy_session_and_data();

    echo htmlspecialchars("Welcome back $forename.<br>
          Your full name is $forename $surname.");

  }
  else echo "Please <a href='authenticate2.php'>click here</a> to log in.";

  function destroy_session_and_data()
  {
    $_SESSION = array();
    setcookie(session_name(), '', time() - 2592000, '/');
    session_destroy();
  }
?>
```

La primera vez que navegues de *authenticate2.php* a *continue.php*, se mostrarán todas las variables de sesión. Pero, debido a la llamada a `destroy_session_and_data`, si luego haces clic en el botón Reload de tu navegador, la sesión se habrá destruido y se te pedirá que vuelvas a la página de inicio de sesión.

Configuración del tiempo de espera

Hay otras ocasiones en las que puedes querer cerrar la sesión de un usuario por tu cuenta, como por ejemplo cuando el usuario se ha olvidado de cerrar la sesión o actuado de forma negligente al hacerlo, y tú deseas que el programa lo haga por él y por su propia seguridad. Esto se hace configurando el tiempo de espera, después del cual se producirá el cierre de sesión automáticamente si no ha habido actividad.

Para ello, utiliza la función `ini_set` de la siguiente manera. Este ejemplo establece el tiempo de espera en exactamente un día:

```
    ini_set('session.gc_maxlifetime', 60 * 60 * 24);
```

Si deseas saber cuál es el periodo de tiempo de espera en un momento determinado, puedes visualizarlo utilizando lo siguiente:

```
echo ini_get('session.gc_maxlifetime');
```

Seguridad de sesión

Aunque he mencionado que una vez que hayas autenticado a un usuario y configurado una sesión, se puede tener la seguridad de que las variables de sesión son confiables, esto no es exactamente así. La razón es que es posible usar el *rastreo de paquetes* (muestreo de datos) para descubrir los ID de sesión que pasan por una red. Además, si se pasa el ID de la sesión en la parte GET de un URL, podría aparecer en los registros externos del servidor del sitio.

La única forma verdaderamente segura de evitar que se descubran es implementar *Transport Layer Security* (TLS, el sucesor más seguro de *Secure Sockets Layer*, o SSL) y ejecutar HTTPS en lugar de páginas web HTTP. Eso queda fuera del alcance de este libro, aunque puede que quieras echar un vistazo a la documentación de Apache (*https://httpd.apache.org/docs/2.4/ssl/*) para ver más detalles de la creación de un servidor web seguro.

Prevención de secuestro de sesión

Cuando SSL no es una posibilidad, puedes autenticar aún más a los usuarios si almacenas su IP junto con sus otros detalles añadiendo una línea como la siguiente cuando guardas sus sesiones:

```
$_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
```

A continuación, como comprobación adicional, siempre que se cargue alguna página y haya una sesión disponible, realiza la siguiente comprobación. Llama a la función `different_user` si la dirección IP almacenada no coincide con la actual:

```
if ($_SESSION['ip'] != $_SERVER['REMOTE_ADDR']) different_user();
```

El código que pongas en tu función `different_user` depende de ti. Yo recomiendo que borres la sesión en curso y pidas al usuario que inicie sesión de nuevo debido a un error técnico o, si tienes su dirección de correo electrónico, envíale un enlace para confirmar su identidad, lo que le permitirá conservar todos los datos de la sesión.

Por supuesto, debes tener en cuenta que los usuarios que se encuentran en el mismo servidor proxy o que comparten la misma dirección IP en una red doméstica o empresarial tendrán la misma dirección IP. De nuevo, si esto es un problema, usa HTTPS. También puedes guardar una copia de la *cadena de agente de usuario* del navegador (una cadena que los desarrolladores ponen en sus navegadores para identificar tipo y versión), con la que podrías también distinguir a los usuarios, debido a la gran variedad de tipos de navegadores, versiones, y plataformas informáticas en uso (aunque esto no es una solución perfecta, y la cadena cambiará si el navegador se actualiza automáticamente). Utiliza lo siguiente para almacenar el agente de usuario:

```
$_SESSION['ua'] = $_SERVER['HTTP_USER_AGENT'];
```

Y usa lo siguiente para comparar la cadena de agente de usuario actual con la cadena guardada:

```
if ($_SESSION['ua'] != $_SERVER['HTTP_USER_AGENT']) different_user();
```

O mejor aún, combina las dos comprobaciones del siguiente modo y guarda la combinación como una cadena hexadecimal `hash`:

```
$_SESSION['check'] = hash('ripemd128', $_SERVER['REMOTE_ADDR'] .
    $_SERVER['HTTP_USER_AGENT']);
```

Y utiliza esto para comparar las cadenas actuales y las almacenadas:

```
if ($_SESSION['check'] != hash('ripemd128', $_SERVER['REMOTE_ADDR'] .
    $_SERVER['HTTP_USER_AGENT'])) different_user();
```

Prevención de fijación de sesión

La *fijación de sesión* ocurre cuando un tercero malintencionado obtiene una ID de sesión válida (que podría ser generada por el servidor) y hace que el usuario se autentique con esa ID de sesión, en lugar de autenticarse con la suya propia. Puede suceder cuando un atacante aprovecha la capacidad de pasar una ID de sesión en la parte GET de un URL, como esta:

```
http://yourserver.com/authenticate.php?PHPSESSID=123456789
```

En este ejemplo, la ID de sesión inventada de 123456789 se pasa al servidor. Ahora, considera el Ejemplo 12-9, que es susceptible de sufrir la fijación de sesión. Para ver cómo, escríbelo y guárdalo como *sessiontest.php*.

Ejemplo 12-9. Una sesión susceptible de sufrir la fijación de sesión

```
<?php // sessiontest.php
  session_start();

  if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;
  else ++$_SESSION['count'];

  echo $_SESSION['count'];
?>
```

Una vez guardado, llámalo en tu navegador mediante el siguiente URL (precédelo con la ruta de acceso correcta, como *http://localhost*):

```
sessiontest.php?PHPSESSID=1234
```

Presiona Reload varias veces y verás que el contador aumenta. Ahora intenta hacerlo con:

```
sessiontest.php?PHPSESSID=5678
```

Presiona Reload unas cuantas veces, deberías verlo contar de nuevo desde 0. Deja el contador en un número diferente al del primer URL, regresa al primer URL y mira cómo cambia el número de nuevo. Has creado dos sesiones diferentes a tu elección, y podrías crear fácilmente tantas como necesitaras.

La razón por la que este enfoque es tan peligroso es que un atacante malicioso podría intentar distribuir estos tipos de URL a usuarios desprevenidos y, si alguno de ellos siguiera estos enlaces, ¡el atacante podría regresar y tomar el control de cualquier sesión que no se haya borrado o haya caducado!

Para evitarlo, cambia el ID de la sesión mediante `session_regenerate_id` tan pronto como puedas. Esta función mantiene todos los valores de las variables de la sesión en curso, pero sustituye la sesión ID con una nueva que un atacante no pueda conocer.

Para ello, busca una variable de sesión especial que te inventes arbitrariamente. Si no existe, ya sabes que se trata de una nueva sesión, así que simplemente cambia el ID de la sesión y configura la variable de sesión especial para anotar el cambio.

El Ejemplo 12-10 muestra cómo podría verse el código para hacer esto con la variable de sesión `initiated`.

Ejemplo 12-10. Regeneración de sesión

```
<?php
  session_start();

  if (!isset($_SESSION['initiated']))
  {
    session_regenerate_id();
    $_SESSION['initiated'] = 1;
  }

  if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;
  else ++$_SESSION['count'];

  echo $_SESSION['count'];
?>
```

De esta manera, un atacante puede volver a tu sitio mediante cualquiera de los ID de sesión que generó, pero ninguno de ellos llamará a la sesión de otro usuario, ya que todos ellos habrán sido reemplazados por ID regenerados.

Forzar sesiones solo con cookies

Si estás preparado para requerir a tus usuarios que habiliten cookies en tu sitio web, puedes utilizar la función `ini_set`, así:

```
ini_set('session.use_only_cookies', 1);
```

Con este ajuste, el truco `?PHPSESSID=` se ignorará. Si utilizas esta medida de seguridad, también te recomiendo que informes a tus usuarios de que tu sitio requiere cookies (pero solo si el usuario tiene desactivadas las cookies), para que sepan lo que está mal si no obtienen los resultados que desean.

Uso de servidor compartido

En un servidor compartido con otras cuentas, no querrás que todos los datos de tu sesión se guarden en el mismo directorio que los de los demás. En lugar de eso, deberías elegir un directorio al que solo tu cuenta tenga acceso (y que no sea visible desde la web) para almacenar tus sesiones; coloca una llamada `ini_set` cerca del inicio de tu programa, así:

```
ini_set('session.save_path', '/home/user/myaccount/sessions');
```

La opción de configuración mantendrá este nuevo valor solo durante la ejecución del programa, y la configuración original se restaurará al final del programa.

Esta carpeta de sesiones puede llenarse rápidamente; es posible que desees eliminar periódicamente las sesiones más antiguas, en función de lo ocupado que esté tu servidor. Cuanto más se usa, menos tiempo querrás mantener una sesión almacenada.

Recuerda que sus sitios web pueden y estarán sujetos a intentos de piratería informática. Hay robots automatizados que ocasionan disturbios en Internet y tratan de encontrar sitios vulnerables para explorarlos. Así que hagas lo que hagas, siempre que estés gestionando datos que no se generen al 100 % dentro de tu propio programa, debes tratarlos siempre con la máxima precaución.

En este punto, deberías tener un buen conocimiento tanto de PHP como de MySQL, así que el próximo capítulo es el momento de introducir la tercera tecnología más importante tratada en este libro, JavaScript.

Preguntas

1. ¿Por qué se debe transferir una cookie al inicio de un programa?
2. ¿Qué función de PHP almacena una cookie en un navegador web?
3. ¿Cómo puedes destruir una cookie?
4. ¿Dónde están almacenados el nombre de usuario y la contraseña en un programa PHP cuando utilizas autenticación HTTP?
5. ¿Por qué la función `password_hash` es una fuerte medida de seguridad?
6. ¿Qué se entiende por *salado* de una cadena?
7. ¿Qué es una sesión PHP?
8. ¿Cómo se inicia una sesión de PHP?
9. ¿Qué es el secuestro de sesión?
10. ¿Qué es la fijación de sesión?

Consulta "Respuestas del Capítulo 12" en la página 714 en el Apéndice A para comprobar las respuestas a estas preguntas.