¿Que es una Cookie?

Una cookie es **información** enviada desde un servidor de páginas web y **almacenada** en el **disco duro** del **visitante** a través del navegador. Esta información será reenviada de nuevo al servidor en cada petición, de forma que el servidor puede identificar o recuperar información sobre el usuario que está accediendo.

¿Por qué se han creado las cookies?

Las cookies fueron implementadas por primera vez por Netscape Communications para la creación del típico cesto de comprar en una tienda online. El problema hasta entonces era que el protocolo HTTP carecía de la posibilidad de mantener información pos sí mismo. Los métodos usados antes eran:

- **Identificación por IP**: un método muy poco fiable, pues bajo una misma IP podían estar accediendo distintos usuarios (por ejemplo desde un cíber), además que la dirección IP de un usuario puede cambiar.
- **Por URL**: Consiste en añadir la información en la URL, despues del interrogante **?**. Esta es una técnica más precisa en lo que se refiere a identificación, pero tiene problemas de seguridad.

Gracias a las cookies, un servidor web puede identificar un conjunto pc-navegador-usuario y mostrar la información adecuada a ese conjunto, por ejemplo un carrito de compra que haya creado.

¿Son inseguras las Cookies?

Existe una tendencia entre la gente a pensar que las cookies son una especie de virus o gusano. Esto es debido a que las cookies están en muchas ocasiones vinculadas a temas de spyware.

La fama, la tendencia a pensar que las cookies son inseguras o malas se ha debido a que en numerosas ocasiones han sido vinculadas a temas de Spyware y otro tipo de situaciones comprometidas para el usuario.

Muchas empresas de publicidad las utilizan para hacer un seguimiento de los hábitos de navegación del usuario. Esto, a mucha gente le parece que es un asalto a la privacidad y por ello las cookies tienen sus detractores.

Sin embargo, el uso de cookies, a día de hoy, y con un navegador actualizado, no suponen ningún tipo de riesgo para el PC ni los datos almacenados en él.

¿Cómo se crea una Cookie?

Vamos a entrar en algo más técnico y ver como se establecen / reciben cookies a través del protocolo HTTP. Cuando entramos en una web, por ejemplo http://www.example.com/index.html, el navegador enviará una petición al servidor parecida a esta:

1. GET /index.html HTTP/1.1

Entonces el servidor hará lo que tenga que hacer para generar el contenido y devolverá algo así:

1. HTTP/1.1 200 OK
2. Content-type: text/html
3. (contenido de la página)

Si desde el servidor se quiere guardar una cookie en el ordenador del usuario, añadirá lo siguiente a su respuesta:

1. Set-Cookie: Name=el valor de la cookie; expires=Fri, 23-Jul-2010 23:59:59 GM T; path=/; domain=.example.com

Vamos a ver por partes qué significa esa línea:

- **Set-Cookie**: Es el identificador que reconocerá el navegador para saber que se va a crear una cookie.
- **Name=valor de la cookie**: Es el conjunto clave valor de la cookie. Name es el nombre identificado, siendo su valor lo que va inmediatamente después del signo igual (=). El valor pueden contener textos y números.
- **expires**: Establece cuando la cookie dejará de existir. Si no se establece, lo cookie ser borrará al cerrar el navegador. Si se establece un tiempo inferior al actual, tambien se borrará.
- domain: Establece el dominio al que la cookie será enviada. Si el valor es por ejemplo, '.example.com', la cookie será enviada a cualquier subdominio existente en example.com. Si se le diese el valor img.example.com, la cookie solo sería enviada a ese subdominio, descartando también al propio example.com .
- **path**: Establece el directorio al que afecta la cookie. Si por ejemplo se establece a '/user', la cookie solo existirá en example.com/user. Es importante destacar que path y domain trabajan de forma conjunta.

Crear y leer cookies en PHP

Vamos a llevar todo esto tan teórico a la práctica usando PHP como lenguaje de programación. En php existe la función **setcookie** para crear o eliminar cookies y la variable global **$_COOKIE** para acceder a su contenido.

La funcion setcookie tiene la siguiente definición:

1. bool setcookie (string $name [, string $value [, int $expire = 0 [, string $path [, s tring $domain [, bool $secure = false [, bool $httponly = false]]]]])

Supongo que puedes vincular fácilmente cada parámetro con los valores que hemos listado en la sección anterior excepto **$secure** y **$httponly**.

- **$secure** establece si la cookie debe solamente funcionar bajo conexiones protegidas por SSL.
- **$httponly** es para indicar si la cookie es solo accesible a través del protocolo HTTP, evitando así el acceso desde lenguajes de scripting como Javascript. Esto puede ayudar a reducir bastante los ataques XSS.

Veamos algunos ejemplos prácticos.

Ejemplo de creación de cookies

```
<?php
$value = 'valor que se quiere guardar';

setcookie("TestCookie", $value, time()+3600);  /* expire en 1 hora */

setcookie("TestCookieDomain", $value, time()+3600,"/","tests.example.com");
```

Accediendo a las cookies

```
<?php
/*Antes de acceder a una cookie, se comprueba si se ha enviado*/

if(isset($_COOKE['TestCookie'])){

    var_dump($_COOKIE['TestCookie']);

}
```

Borrando cookies

```
<?php
/*Tiempo de expiración pasado, borra la cookie*/

setcookie ("TestCookie", "", time() - 3600);
```

Apuntes sobre seguridad

Hay que tener mucho cuidado cuando se crean o leen cookies ya que existen muchos ataques que se aprovechan de excesos de confianza / desconocimiento del programador.

- **Nunca almacenes información privada en una cookie**: Un ejemplo muy común de uso de cookies es la creación de sesiones con la opción “Recordarme”. En esta clase de webs, el usuario es recordado y no tiene que volver a escribir sus credenciales para iniciar sesión. Si estás pensando en almacenar el usuario y la contraseña en las cookies, vas por mal camino, pues es fácil acceder a la información de las cookies a través de un sniffer u otros ataques. Para que no os quedéis con la duda, lo que se suele hacer es generar un código aleatorio que identifique la sesión del usuario.
- **Nunca te fíes del contenido de una cookie**: Si bien las cookies funcionan de forma totalmente transparente al usuario y no tiene, a simple vista, formas de modificarlas, no te puedes fiar del contenido que envíen. Existen múltiples herramientas que permiten modificar / crear las cookies de forma que un usuario

avanzado podría hacer grandes destrozos en el servidor. Valida siempre el valor de las cookies para comprobar que tienen lo esperado y no, por ejemplo, inyección de código.

¿Que es una sesión?

Un sesión en una web se puede definir como el recorrido de páginas que un usuario hace en nuestro sitio, desde que entra hasta que lo abandona. Gracias al uso de sesiones podemos reconocer las peticiones de cada usuario y así llevar a cabo acciones específicas, como mostrar información adaptada a él o guardar información de sus gustos o páginas que más visita.

Podemos entonces decir que pueden diferenciarse dos tipos de sesiones:

- **Sesiones "activas":** las que muestran información personalizada según el usuario, por ejemplo, cuando inicia sesión en una web.
- **Sesiones "pasivas":** el servidor reconoce cada movimiento del usuario y lo almacena de forma que en un futuro, se le mostrará una web con la información que al usuario le pueda parecer más interesante sin que se dé cuenta.

Lógicamente, una web puede incorporar los dos tipos de sesiones, véase el caso de los servicios ofrecidos por Google .

¿Cómo funcionan?

Las sesiones se basan en un identificador generado por visitante, simulando una especie de DNI, de modo que cuando quiera acceder a cualquier página de nuestro sitio, mostrará ese "DNI" y quedará identificado. El identificador será único por cada usuario y se le conoce como **Session ID** (SID).

Ahora la pregunta es **¿Como se gestiona dicho SID?** pues muy fácil, con Cookies. Si no sabes muy bien que son, te recomiendo mi anterior artículo Introducción a Cookies en la Web. Una cookie es información que se establece desde el servidor y que el navegador del usuario envía en cada petición de forma abstracta. Entonces, si establecemos en una cookie el identificador de sesión, ya tenemos nuestra tarjeta identificadora lista.

Desde el lado del servidor, se debe implementar toda la lógica de la sesión, almacenando todos los SID activos y la información relativa a ellos. Esto se puede hacer de múltiples formas: mediante sistema de ficheros, base de datos, memcached o cualquier otro método de almacenamiento de información.

Manejo de sesiones en PHP

PHP, como es de esperar tiene todo un set de funciones para el manejo de sesiones de forma que abstrae bastante la labor del desarrollador. Entre otras cosas, llevará a cabo las siguientes tareas:

- Identificación por cookie del SID.
- Si no existe el SID, crea uno y lo guarda en las cookies.

- Almacenamiento de información relativa al SID en el sistema de ficheros.
- Gestión del expirado de sesión.

Almacenar y/o recuperar información sobre la sesión actual es tan fácil como usar un array global llamado **$_SESSION**.

Con todo esto, el uso de sesiones en PHP se hace realmente fácil. Aunque los creadores de PHP no nos quieren encadenar y nos permiten crear nuestras propias funciones para gestionar las sesiones de forma manual, como veremos más adelante en un ejemplo.

Usando los manejadores de PHP

Vamos a mostrar un ejemplo de como crear una sesión por visitante y almacenar información persistente durante la sesión

view plaincopy to clipboardprint?

```
1. <?php
2. //Inicia o recupera la sesión
3. session_start();
4. //Ejemplo de como crear una variable de sesión
5. if(!isset($_SESSION['user_name'])){
6.     $_SESSION['user_name'] = getUserName();
7. }
8. //Actualiza o crea una variable de sesión
9. $_SESSION['last_access'] = time();
```

Es importante informar sobre la función **session_start**. Dicha función se encarga de crear o cargar una sesión previamente abierta basándose en el SID pasado por cookies. Algo a tener muy en cuenta, es que al estar basado en cookies, el server generará cabeceras HTTP, por lo que es importante llamar a la función antes de que se envíe cualquier salida de texto.

Si no recibirás un error que dice algo como **“headers already sent”**. De hecho os recomiendo que pongáis la llamada en la primera línea de vuestro index.php para evitar problemas.

Usando manejadores propios

Como hemos dicho, PHP nos permite declarar nuestras propias funciones para gestionar las sesiones de forma personalizada, a continuación os dejo una clase base que podéis rellenar para crear vuestros manejadores:

view plaincopy to clipboardprint?

```
1. <?php
```

```
2.  /**
3.  * Manejador de sesiones
4.  *
5.  */
6.  class Session
7.  {
8.  /**
9.  * Abre la sesión
10. * @return bool
11. */
12. public static function open() {
13. }
14.
15. /**
16. * Cierra la sesión
17. * @return bool
18. */
19. public static function close() {
20. }
21.
22. /**
23. * Lee la sesión
24. * @param int session id
25. * @return string contenido de la sesión
26. */
27. public static function read($id) {
28. }
29.
30. /**
```

```
31. * Guarda la sesión
32. * @param int session id
33. * @param string contenido de la sesión
34. */
35. public static function write($id, $data) {
36. }
37.
38. /**
39. * Destruye la sesión
40. * @param int session id
41. * @return bool
42. */
43. public static function destroy($id) {
44. }
45.
46. /**
47. * Colector de basura
48. * @param int life time (sec.)
49. * @return bool
50. */
51. public static function gc($max) {
52. }
```

Una vez que tengas la clase adaptada a tus necesidades, necesitas indicarle a PHP que use eses métodos y no los que usa por defecto.

Eso se consigue con la función **session_set_save_handler**:

view plaincopy to clipboardprint?

```
1. ini_set('session.save_handler', 'user');
2. session_set_save_handler(array('Session', 'open'),
3.                          array('Session', 'close'),
4.                          array('Session', 'read'),
```

```
5.                  array('Session', 'write'),
6.                  array('Session', 'destroy'),
7.                  array('Session', 'gc')
8.                  );
9.  session_start();
```

Configuración de sesiones en PHP

En la documentación oficial, podéis acceder a esta página en la que se muestra un listado completo de todas las variables de configuración (en el fichero php.ini), no obstante, vamos a ver algunas que tienen bastante importancia.

session.save_path

Especifica en que directorio será almacenda la informacion de los distintos SID generados. En este directorio tiene que poder escribir el usuario de Apache, que será el creador de los ficheros en última instancia.

session.name

Especifica el nombre de la sesión por defecto, que será a su vez el nombre de la cookie establecida por el servidor. Por defecto se llama PHPSESSID pero es recomendable cambiarlo.

session.gc_maxlifetime

Es el número de segundos tras el cual la información almacenda pasa a ser considerada basura y por tanto borrada cuando se lance el colector de basura (Garbage Collector).

session.cookie_lifetime

Establece los segundos durante los cuales la cookie de sesión va a estar activa. Por defecto está a 0 y por tanto la cookie de sesión durará hasta que el usuario cierre el navegador, como ya habíamos visto en Introducción a Cookies en la Web.

Es muy importante diferenciar esta variable de **gc_maxlifetime**. La funcionalidad de **gc_maxlifetime** hace la resta de la hora del último acceso menos la hora actual, si el resultado en segundos es mayor que el valor de dicha variable, la sesión se borra.

Mientras que **cookie_lifetime** establece un tiempo de duración fijo, es decir, si se establace en 60 segundos, la sesión expirará dentro de un minuto aunque el usuario esté accediendo continuamente.

session.use_only_cookies

Esta variable establece que solamente se deben de usar cookies para gestionar las sesiones. Por defecto está activado y **es importante no modificarlo**. El problema de

desactivarlo, es que PHP intentará gestionar las sesiones vía parámetros GET en caso de que no estén activadas las cookies en el navegador del usuario.

Esto puede llevar a problemas de seguridad y comprometer la privacidad del usuario.