Guion para la Unidad 4

Sitio: Centros - Granada
Curso: Desarrollo web en entorno servidor
Libro: Guion para la Unidad 4

Imprimido por: Aguilera Aguilera, Javier
Día: miércoles, 17 de noviembre de 2021, 08:48

Tabla de contenidos

1. Cookies

Como HTTP es un protocolo sin estado, ninguna petición guarda relación con las anteriores o posteriores y, surge el problema de identificar a los usuarios.

Por ejemplo, cuando deseamos identificarlo para mostrarle un contenido privado, de tal forma que otro usuario no pueda acceder sin más a esa URL privada.

Como solución surgieron las cookies.

Las cookies son pequeños ficheros que dejan los servidores web en los ordenadores de los clientes.

En general se usan para identificar usuarios y personalizar las páginas. Por ejemplo, para identificar a clientes anónimos en nuestro carrito de la compra, mejorar experiencias de otros usuarios, recordar idiomas, almacenar información sobre la fecha de la última visita, etc.

Cuando un cliente realiza una petición web, envía al servidor las cookies que pudiera tener de este. (Sólo se envían al servidor que las emitió, no a otros). Pero, evidentemente, esto perjudica la privacidad de los usuarios ya que una cookie nos puede rastrear en cada momento.

Las cookies constan de un nombre y un valor. Además, pueden tener fecha de caducidad. Si no se indica la fecha de caducidad se borran al cerrar el navegador (cookies de sesión) .

Para manejar las cookies se usa la función setcookie()que tiene el siguiente formato:

setcookie(string $name,string $value = "",int $expires = 0,string $path = "",string $domain = "",bool $secure = **false**,bool $httponly = **false**): bool

```
(La fecha en la que expira la cookie se especifica como una fecha Unix, es
decir, el número de segundos pasados desde el comienzo de 1970. Normalmente se
usa la función time(), que devuelve la fecha actual y se le suma un periodo de tiempo expresado en segundos)
```

Las cookies se envían como cabeceras en la peticiones HTTP.

Recuerda que hay que enviar las cabeceras antes de empezar con el cuerpo de la respuesta. Esto implica que hay que utilizar la función setcookie() antes de que se empiece a escribir la salida. Si, por ejemplo, intentamos llamar a setcookie() después de haber realizado un echo se producirá un error.

Siempre que utilices cookies en una aplicación web, debes tener en cuenta que, en última instancia, su disponibilidad está controlada por el cliente. Por ejemplo, algunos usuarios deshabilitan las cookies en el navegador porque piensan que la información que almacenan puede suponer un potencial problema de seguridad. O la información que almacenan puede llegar a perderse porque el usuario decide formatear el equipo o simplemente eliminarlas de su sistema.

2. Sesiones

Como HTTP es un protocolo sin estado, las diferentes peticiones de un cliente al servidor son independientes, no están relacionadas entre sí. Es decir, cada página web es un documento independiente, lo que hace que dos programas PHP no puedan, en principio, compartir información (utilizar variables comunes sin que la información salga del servidor). Para asociarlas, se utilizan las **sesiones**.

PHP nos ofrece, mediante las variables de sesión, un mecanismo para almacenar variables en un espacio accesible en las distintas peticiones de un usuario. Este espacio de memoria está vinculado al usuario que realiza la petición, y no a la petición en sí. A este espacio lo llamamos sesión.

La información guardada en una sesión puede llamarse en cualquier momento mientras la sesión esté abierta. Se puede cerrar por intervención del usuario o, por ejemplo, pasado un tiempo.

Al iniciar una sesión el servidor asigna y envía al usuario un identificador de sesión único. En las siguientes peticiones el usuario envía al servidor ese identificador, de manera que el servidor sabe que se trata del mismo usuario. Esto podemos usarlo para hacer persistente la información de estado entre peticiones de páginas.

Para controlar la sesión el servidor deja, automáticamente, una cookie (de sesión) con el ID de sesión (número aleatorio único) en el cliente, que se elimina al cerrarla. Si se borran manualmente las cookies del navegador, se cierran las sesiones abiertas. Se puede configura el servidor para contralar las sesiones sin cookies, pero no es lo habitual. La ausencia de un ID o una cookie de sesión permite a PHP crear una nueva sesión y generar un nuevo ID de sesión.

Las sesiones sirven, por tanto, para que persista la información mientras el usuario está navegando y poder compartir datos entre todas las páginas de nuestra aplicación web (por ejemplo, mientras tenemos nuestro carrito de la compra). Dicho de otra forma, un usuario puede ver varias páginas durante su paso por un sitio web y con sesiones podemos almacenar variables a las que podremos acceder en cualquiera de esas páginas.

En la sesión de un usuario podemos almacenar todo tipo de datos, como su nombre, productos de un hipotético carrito de la compra, preferencias de visualización o trabajo, páginas por las que ha pasado, etc. Todas estas informaciones se guardan en lo que denominamos variables de sesión.

Los datos se almacenan en el servidor y son invisibles al cliente web.

Para crear una sesión se utiliza la función session_start() que es la que envía al usuario la cookie que lo identificará.

Si no hay una sesión activa la crea, si la hay, en el script que llama a la función se une a ella (es decir, continúa con la sesión que pudiera tener abierta en otra página).

Una vez inicializada la sesión con session_start() podemos utilizar variables de sesión, es decir, almacenar datos para ese usuario que se conserven durante toda su visita o recuperar datos almacenados en páginas que haya podido visitar.

La sesión se tiene que inicializar antes de escribir cualquier texto en la página. De no hacerlo así corremos el riesgo de recibir un error, porque al iniciar la sesión se deben leer las cookies del usuario (esto no se puede hacer si ya se han enviado las cabeceras del HTTP).

Una vez creada la sesión es posible utilizar la variable superglobal $_SESSION para compartir información entre los scripts que compartan sesión. En un array asociativo se almacenan las variables de sesión definidas por el usuario al que podemos añadir elementos:

```
$_SESSION[“nombre”]=valor;
```

En esta variable de sesión no podemos guardar los recursos (como, por ejemplo, un fichero).

Si queremos cerrar una sesión tenemos que destruir la matriz $_SESSION y el identificador de la sesión.

Las sesiones se pueden cerrar de varias maneras:

- El usuario puede cerrar la sesión simplemente cerrando el navegador (no basta con cerrar las pestañas).
- Un programa puede cerrar la sesión mediante la función session_destroy().
- En general, las cookies tienen una duración establecida en la directiva session.cookie_lifetime (y el servidor puede borrar la información cuando ha pasado el tiempo indicado en segundos en la directiva session.gc_maxlifetime), pero la duración de una sesión en particular puede establecerse en el momento de su creación mediante la función session_set_cookie_params() (tiempo que se puede modificar posteriormente).

Cuando se destruye una sesión, el programa que ha destruido la sesión sigue teniendo acceso a los valores de $_SESSION creados antes de la destrucción de la sesión, pero las páginas siguientes no.

Ejemplos:

- https://www.w3schools.com/php/php_sessions.asp
- Otro ejemplo.

3. Diferencias entre sesiones y cookies

- Las sesiones finalizan al cerrar el navegador
- Las cookies pueden finalizar a largo plazo
- Las sesiones son seguras ya que el usuario solo tiene acceso a su ID de sesión, la información se guarda en el servidor. Esta información se crea y se mantiene en el servidor hasta que se cierra la sesión. Esto puede ocurrir por intervención del usuario o por tiempo de expiración designado.
- Las cookies pueden ser inseguras puesto que se guardan en el cliente. No son recomendadas para almacenar información sensible puesto que es información enviada por el cliente (y puede ser alterada por terceros). Cualquier dato importante debe ser siempre tratado con la mayor seguridad posible.

4. Tiempo de vida de una sesión

Si un usuario inicia sesión en un lugar y se olvida de cerrarla, otros pueden continuar con la misma, por eso es mejor establecer un tiempo máximo de sesión.

El siguiente código asegura que si no hay ninguna actividad en 10 minutos, cualquier request en adelante redigirirá a la página de logout.

```php
session_start();
// Establecer tiempo de vida de la sesión en segundos
$inactividad = 600;
// Comprobar si $_SESSION["timeout"] está establecida
if(isset($_SESSION["timeout"])){
    // Calcular el tiempo de vida de la sesión (TTL = Time To Live)
    $sessionTTL = time() - $_SESSION["timeout"];
    if($sessionTTL > $inactividad){
        session_destroy();
        header("Location: /logout.php");
    }
}
// El siguiente key se crea cuando se inicia sesión
$_SESSION["timeout"] = time();
```

Fuente:https://diego.com.es/sesiones-en-php

5. Propagar el SID

Como ya hemos visto, una sesión se crea en el momento en que un usuario entra en un sitio web, pudiéndose identificar o acceder anónimamente. Al entrar, se genera un identificador (ID), que puede seguirse mientras esté conectado y de esta forma saber qué informaciones obtiene o qué operaciones realiza el citado usuario. Los datos de la sesión asociados a este identificador se guardan en el servidor dentro de un fichero en el directorio especificado en la sección [Session] del fichero de configuración php.ini.

Hay dos maneras de propagar el ID del usuario entre las diferentes páginas :

- Mediante Cookies (se recomienda encriptarlo)
- A través de Parámetros URL. El identificador se añade como una parte más de la URL, de la forma: http://www.misitioweb.com/tienda/listado.php&PHPSESSID=34534fg4ffg34ty ;

  (En el ejemplo anterior, el SID es el valor del parámetro PHPSESSID)

Lo más habitual es hacerlo mediante cookies. Este identificador tiene un tiempo de vida y, por tanto, la sesión que lo ha creado también caduca. Este tiempo puede fijarse en el fichero php.ini dentro de la variable session.cookie_lifetime . Por defecto el valor está establecioda 0 para que caduque cuando se cierre el navegador. Si se establece otro valor, la sesión caducará al pasar esos segundos y no es necesario cerrar el navegador.

6. Soporte para sesiones

En el manual puedes encontrar toda la información sobre el manejo de sesiones. En concreto:

- Directivas para configurar el manejo de sesiones
- Pasar ID de sesión
- Constantes predefinidas
- Funciones de sesión

7. Sesiones y seguridad

Una de las principales preocupaciones de los desarrolladores de software es la posibilidad de que algún hacker pueda quebrantar la seguridad de nuestra aplicación y robar información que los usuarios nos han confiado.

Si esto sucede nuestros clientes verían comprometidas sus cuentas y nosotros estaríamos en serias dificultades.

Una de las formas en las que un atacante puede colarse en nuestros sistemas son las sesiones.

Como ya hemos visto, toda la información de las sesiones está almacenada del lado del servidor, por lo que los riegos se reducen a:

- Que una persona no autorizada pueda acceder directamente a los archivos del servidor.
- Que un atacante suplante la identidad de un usuario que ha iniciado sesión

Una forma de atacar una aplicación web a través del mecanismo de sesiones consiste en el *secuestro de la sesión* (session hijacking). El atacante roba una sesión ID, copia el contenido de una cookie de sesión y crea con él otra cookie en otro ordenador. Después tratará de usarla para entrar en la aplicación como si fuese el verdadero usuario.Si no se han tomado medidas de precaución, el servidor no será capaz de distinguir a un visitante del otro.

Obtener una cookie de sesión no es algo sencillo, pero tampoco imposible. En realidad, basta con:

- Tener acceso al ordenador de la víctima
- Interceptar el tráfico de red (si no esté cifrado)
- Ejecutar un ataque de tipo XSS

Por lo tanto, es muy importante que el mecanismo de validación de identidad de un visitante.

En el tutorial de PHP tenemos más información sobre sesiones y seguridad y sobre las configuraciones que se deben adoptar.

En general, hay algunas recomendaciones que son básicas como, por ejemplo:

- Usar siempre HTTPS con lo que usaremos tráfico cifrado.
- Cambiar el nombre de tu cookie de sesión para que sea más difícil saber cuál es la cookie que contiene el ID de sesión. Por defecto su nombre es PHPSESSID. Puedes cambiarlo usando session_name() o desde la configuración de php (variable session.name).
- Habilitar el modo de sesión estricto (por defecto deshabilitado). Así el servidor sólo reconocerá como válidas aquellas sesiones cuyo identificador haya sido generado por él. Para ello modifica la entrada session.use_strict_mode del archivo php.ini.
- Forzar el uso de cookies de sesión. Con esto evitamos que alguien, sin darse cuenta, comparta un enlace que contiene su ID de sesión. Basta con modificar el valor de session.use-only-cookies
- Cuando inicialices una sesión no guardes sólo el ID de usuario. Guarda también algún otro dato que permita validar la identidad del visitante.

Ejemplo:

```
<?php
session_start();
if (successfulLogin()){
$_SESSION['uid'] = $userId;
$_SESSION['userAgent'] = $_SERVER['HTTP_USER_AGENT'];
$_SESSION['ip'] = $_SERVER['REMOTE_ADDR'];
}
```

Después, en cada página donde se deba validar la identidad del usuario:

```
<?php
session_start();
if (empty($_SESSION['uid']) || isValidUser($_SESSION['uid'])) {
http_response_code(403);
die;
}
if ($_SESSION['userAgent'] !== $_SERVER['HTTP_USER_AGENT']) {
http_response_code(403);
die;
}
if ($_SESSION['ip'] !== $_SERVER['REMOTE_ADDR']) {
http_response_code(403);
die;
}
```

De esta forma, si alguien logra robar la cookie de sesión no podrá utilizarla.

Este artículo, acerca de la seguridad en sesiones, también es interesante : https://diego.com.es/seguridad-de-sesiones-en-php

8. Autenticación HTTP con PHP

Consulta los apuntes de esta unidad y el manual para saber más sobre autenticación HTTP con PHP.

Prueba alguno de los ejemplos propuestos y muestra el resultado en clase

9. Seguridad usuarios y roles

Muchas veces es importante verificar la identidad de los dos extremos de una comunicación.

Los sitios web que necesitan emplear identificación del servidor, como las tiendas o los bancos, utilizan el protocolo HTTPS. Este protocolo requiere de un certificado válido, firmado por una autoridad confiable, que es verificado por el navegador cuando se accede al sitio web. Además, HTTPS utiliza métodos de cifrado para crear un canal seguro entre el navegador y el servidor, de tal forma que no se pueda interceptar la información que se transmite por el mismo. Para identificar a los usuarios que visitan un sitio web, se pueden utilizar distintos métodos como el DNI digital o certificados digitales de usuario, pero el más extendido es solicitar al usuario cierta información que solo él conoce: la combinación de un nombre de usuario y una contraseña.

Así, puede haber sitios web en los que los usuarios autentificados pueden utilizar sólo una parte de la información. Otros sitios web necesitan separar en grupos a los usuarios autentificados, de tal forma que la información a la que accede un usuario depende del grupo en que éste se encuentre. Por ejemplo, una aplicación de gestión de una empresa puede tener un grupo de usuarios a los que permite visualizar la información, y otro grupo de usuarios que, además de visualizar la información, también la pueden modificar.

Debemos distinguir la autentificación de los usuarios y el control de acceso, de la utilización de mecanismos para asegurar las comunicaciones entre el usuario del navegador y el servidor web. Aunque ambos aspectos suelen ir unidos, son independientes.

Para comprobar si un servidor dispone de HTTPS podemos hacerlo así:

```php
if (!isset($_SERVER['HTTPS']) || $_SERVER['HTTPS'] != 'on') {
 // NO DISPONE DE CONEXIÓN HTTPS
}
```

10. Mapa conceptual

Mapa Conceptual