Aplicacións Web con PHP

Sumario

Sesións

Unha sesión é un bloque de información, gardado no servidor, que almacena todo tipo de variables e valores que garda información sobre os usuarios e as páxinas que visitan no noso sitio web, desde que entran no sitio ata que o abandonan.

A sesión créase no momento que o usuario entre no sitio web: ao crear a sesión asígnaselle un identificador (***Session ID*** ou **SID**) en forma de cadea de caracteres, para asociala ao usuario. Este **SID** debe ser propagado de unha a outra páxina no noso sitio web, sempre que o usuario cambia de páxina dentro do noso sitio. Así, na nova páxina pódese recuperar a sesión correspondente.

session_start( ). Inicialización das sesións

Para empregar as sesións primeiro hai que iniciar unha sesión. Isto faise coa función session_start( ). é moi importante que sexa a primeira liña da páxina .php, para que non produza erros. Se queremos evitar que se mostren warnings ou erros podemos empregar a @session_start() (en PHP a arroba antes dunha función evita que se mostren por pantalla warnings e erros).

Exemplo:

```
<?php
@session_start();
echo "Sesión iniciada.";
?>
<!DOCTYPE html>
<html>
...
```

Lembra que ten que ser a primeira liña, sen que existan liñas en branco antes.

Para acceder ao valor do sid, podemos empregar a función session_id():

```
<?php
@session_start();
echo "Sesión iniciada.";
echo " A sesión ten un id de ".session_id();
?>
<!DOCTYPE html>
<html>
...
```

Se temos varios ficheiros enlazados a sesión será a mesma, pois ***session_start( )*** crea unha sesión nova, ou mantén a que está aberta.

Podemos cambiar entre as dúas páxinas e sempre teremos a mesma sesión.

$_SESSION. Array asociativo

Unha vez creada unha sesión, temos dispoñible un array asociativo ***$_SESSION,*** que está dispoñible para o usuario en concreto que iniciou a sesión. Neste array podemos gardar variables que estarán dispoñibles mentres estea aberta a sesión. Por exemplo, podemos dispoñer do nome de usuario ou calquera outra variable en diferentes páxinas enlazadas na mesma sesión:

sesion1a.php

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title></title>
</head>
<body>
<br/>
<!-- DEFINIMOS UNHA VARIABLE -->
<?php
$_SESSION[‘usuario’]=”Xan”;
?>
<h2>Estou na páxina 1a!!  </h2>
<a href="sesion1b.php">Ir a sesion1b </a>

</body>
</html>
```

sesion1b.php

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title></title>
</head>
<body>
<br/>
<?php
/* PODEMOS ACCEDER Á VARIABLE */
echo "O usuario é ”,$_SESSION[‘usuario’];
?>
<h2>Estou na páxina 1b!!  </h2>
<a href="sesion1a.php">Ir a sesion1a</a>
<br>
</body>
</html>
```

Deste xeito poderemos manter no array asociativo $_SESSION variables que permanecen accesibles mentres teñamos activa a sesión, por exemplo, un usuario, un carriño da compra, etc.

Falta por ver como dar seguridade a estas sesións…

session_destroy( ). Rematando a sesión

Para rematar a sesión, e que non estea activa, emprégase a sentencia ***session_destroy()***. Esta non elimina o array asociativo asociado, así que se recomenda borralo explicitamente:

```
<?php
   session_start( );    /*MANTEMOS  A SESIÓN INICIADA  */
   $_SESSION =  array( );   /* ELIMINAMOS TODAS AS VARIABLES */
   // E FINALMENTE DESTRUÍMOS A SESIÓN:
   session_destroy( );
?>
```

session_unset( ). Borrando variables de $_SESSION

Outro xeito de borrar é empregar a función session_unset( ):

```
<?php
   session_start( );    /*MANTEMOS  A SESIÓN INICIADA  */
   session_unset()  /* ELIMINAMOS TODAS AS VARIABLES */
   // E FINALMENTE DESTRUÍMOS A SESIÓN:
   session_destroy( );
?>
```

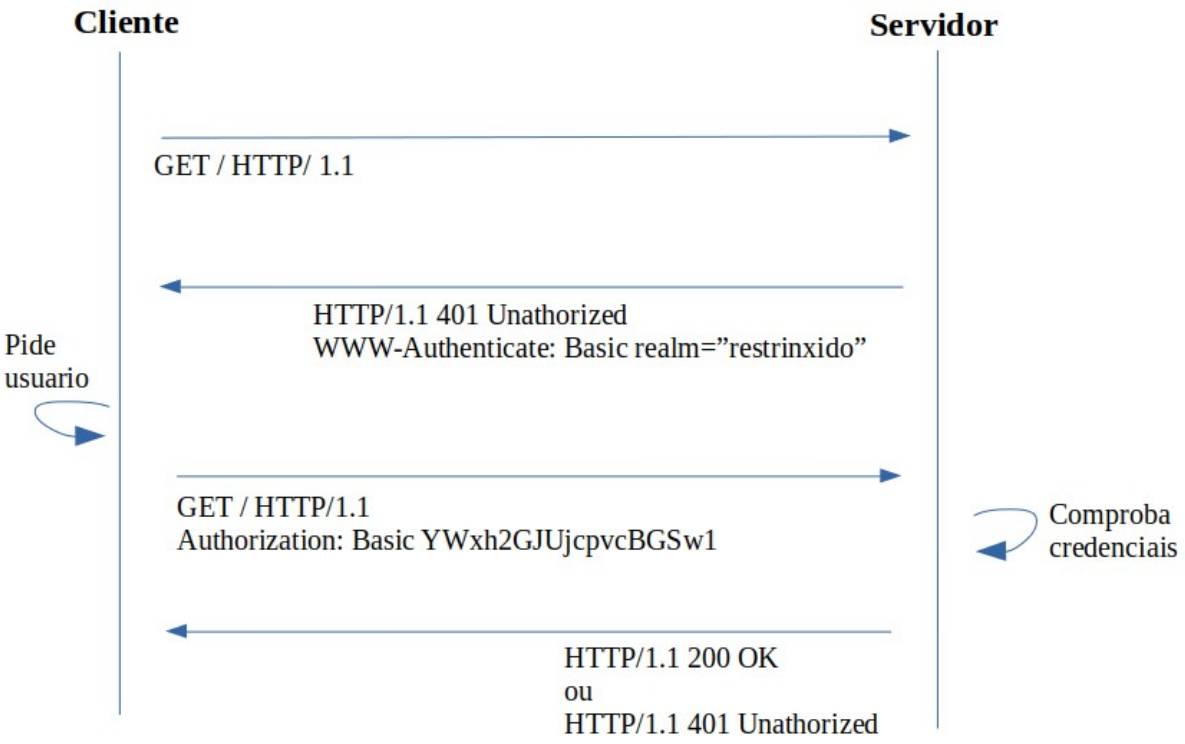
Será conveniente ter un ficheiro ***pecharSesion.php*** ou algo parecido para pechar a sesión, que nos leve a login.php:

```
<?php
   session_start( );    /*RETOMAMOS A SESIÓN INICIADA, QUE QUEREMOS PECHAR */
   $_SESSION =  array( );

   session_destroy( );

   header("Location:... ");
?>
```

Autenticación de usuarios e control de acceso

Autenticación http (WWW-Authenticate)

O protocolo http ofrece un método sinxelo para autenticar os usuarios, baseados no código de estado ***401 (Unathorized)*** e no header ***WWW-Authenticate***:



O navegador envía unha petición GET ao servidor, e a páxina solicitada ten o encabezamento HTTP WWW-Authenticate. O navegador cando recibe ese encabezamento mostra un formulario de usuario/contrasinal, e estes datos son enviados de volta ao servidor, que comproba credencias. Se as credenciais son válidas, envíase a páxina solicitada, se non envíase unha mensaxe de erro Non authorized.

En PHP as credenciais grávanse nas variables de servidor:

- **$_SERVER['PHP_AUTH_USER']** : o nome de usuario introducido
- **$_SERVER['PHP_AUTH_PW']** : contrasinal introducida
- **$_SERVER['AUTH_TYPE']**: o método http empregado para autenticar. Pode ser "basic" ou "digest".

¿Como temos que escribir en PHP a nosa páxina para que o navegador pida usuario e contrasinal? Deste xeito:

//Exemplo1

```
<?php

if (!isset($_SERVER['PHP_AUTH_USER'])) {
        header('WWW-Authenticate: Basic realm="Acceso restrinxido"');
        header('HTTP/1.0 401 Unauthorized');
        echo 'Requerida autenticación para acceder a esta páxina.';
        exit;
}
else {
echo "<p>Introduciches como nome de usuario: {$_SERVER['PHP_AUTH_USER']}.</p>";
echo "<p>Introduciches como contrasinal: {$_SERVER['PHP_AUTH_PW']}</p>";
}
```

Tamén poderíamos facer as comprobacións necesarias para determinar se as credenciais son válidas. Por exemplo, cun usuario 'proba' e contrasinal 'abc123.':

//Exemplo2

```
<?php
if ((!isset($_SERVER['PHP_AUTH_USER'])) ||
      ($_SERVER['PHP_AUTH_USER'] != "proba") || ($_SERVER['PHP_AUTH_PW'] != "abc123"))
    {
    header('WWW-Authenticate: Basic realm="Acceso restrinxido"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Requerida autenticación para acceder a esta páxina.';
    exit;
}
?>
<html>
      <head>
           <title>Exemplo de autenticación http</title>
      </head>
      <body>
           Conseguiu o acceso a zona restrinxida</B>.
      </body>
</html>
```

OLLO: Lembra que http envía a información sen cifrar. Para engadir seguridade ao mecanismo de autenticación http temos que traballar SEMPRE con **https/ssl**

Distintos usuarios en ficheiro

No exemplo anterior un único usuario tiña acceso á parte restrinxida. Nas aplicacións reais precisamos varios usuarios, que podemos gardar nunha base de datos, ou nun ficheiro.
Se empregamos un ficheiro de texto, por exemplo, passwords.txt , podemos darlle un formato: nome_usuario,contrasinal. Así:

passwords.txt

Ana,abc123.
Xan,123456
Miguel,aabbcc

Agora pedimos a autorización ao comezo da páxina se non estamos xa cun usuario permitido. Comprobamos no ficheiro de contrasinais que chamamos *passwords.txt* se o usuario e contrasinal coinciden, e se é así damos acceso á parte restrinxida.
Para ler o ficheiro empregamos a función *file()* de PHP, que transfire un ficheiro completo a un array, cada fila como un elemento do array $fich:

```
 <?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
     header('WWW-Authenticate: Basic realm="Acceso restrinxido"');
     header('HTTP/1.0 401 Unauthorized');
     echo 'Authorizacion requerida.';
     exit;}
 else {
     $fich = file("passwords.txt");
     $i = 0;
     $validado = false;
     while (!$validado && $i < count($fich)) {
          $campo = explode(",", $fich[$i]);
          if (($_SERVER['PHP_AUTH_USER'] == $campo[0]) && ($_SERVER['PHP_AUTH_PW'] ==
               rtrim($campo[1])))
            {
            $validado = true;
            }
          $i++;
     }
     if (!$validado) {
          header('WWW-Authenticate: Basic realm="Acceso restrinxido"');
          header('HTTP/1.0 401 Unauthorized');
          echo 'Authorizacion Requerida.';
          exit;     }
 else {
?>
          <!DOCTYPE html>
          <html lang="es">
              <head>
                    <meta charset="UTF-8" />
```

```
                    <title>Autentiación http</title>
               </head>
               <body>
                    Conseguiu o acceso á zona restrinxida</B> co usuario
                        <?php echo $_SERVER['PHP_AUTH_USER'] ?>.
          </body>
          </html>
          <?php
     }
}?>
```

Autenticación con htpasswd

Apache ten unha utilidade en liña de comando **htpasswd,** que permite almacenar un ficheiro diferentes contrasinais, que se almacenarán cifradas. (podes ver https://httpd.apache.org/docs/2.4/es/howto/auth.html). O ficheiro é conveniente que estea nunha carpeta non accesible vía web.

Creamos ficheiro **.htpasswd** (podería ser calquera outro nome) na carpeta ***/etc/apache2*** e engadimos usuario proba con:

```
htpasswd -c /etc/apache2/.htpasswd proba
```

Isto creará o ficheiro (-c), e nos pedirá un password por consola. Poderíamos engadir máis usuarios sen o modificador -c:

```
htpasswd /etc/apache2/.htpasswd xan
```

Agora teremos que indicar no directorio no que teñamos os contidos restrixidos cales son os usuarios: indicaremos onde está o ficheiro co usuarios e contrasinais. Para isto temos que crear no noso directorio restrinxido un ficheiro ***.htaccess*** coas directivas de Apache seguintes:

```
AuthName "restrinxindo"
AuthType Basic
AuthUserFile /etc/apache2/.htpasswd
require valid-user
```

Ademais hai que asegurarse de que na configuración de Apache (en apache2.conf) utilízase a directiva `AllowOverwrite` para que se aplique correctamente a configuración dos ficheiros .htaccess.

Deste modo Apache só permitirá o acceso á carpeta cos nosos contido restrinxidos, se o usuario e contrasinal está no ficheiro /etc/apache2/.htpasswd.

Autenticación con bases de datos

A autenticación máis avanzada será empregando bases de datos. Os nosos usuarios estarán gardados na base de datos, cun usuario e contrasinal, e para loguearse na nosa aplicación debemos comprobar que son correctos. O contrasinal debe estar cifrado, polo que veremos previamente como é o cifrado de bases de datos

Cifrado de contrasinais na Base de datos con PHP

Os contrasinais dos usuarios na base de datos deberanse gardar cifrados, para que calquera acceso indebido á mesma non proporcione todos os contrasinais dos nosos usuarios: gardaremos en vez do contrasinal en texto plano o resultado de pasarlle unha función de ***hash*** a dito contrasinal.
Despois, cando comprobemos o contrasinal tecleado teremos que comprobar que o resultado de pasar a función de hash ao contrasinal tecleado é o mesmo que o gardado na base de datos.
As funcións básicas/antigas de hashing en php eran as seguintes (pasámoslle un string e nos devolve o string hasheado):

- **md5**

```
string md5 (string $str [, bool $raw_output = false ])

Exemplo:  $contrasinalHashMd5= md5($contrasinalTecleado);
```

Calcula un hash co algoritmo md5. Se se establece _$raw*output* como true devolverá un raw binario cunha lonxitude de 16. Por defecto un hash de 32 caracteres hexadecimal.

- **sha1**

```
string sha1 (string $str [, bool $raw_output = false ])
```

Calcula un hash con el algoritmo sha1. Se se establece _$raw*output* como true devolverá un raw binario con una longitud de 20. Por defecto un hash de 40 caracteres hexadecimal.

- **hash**

```
string hash ( string $algoritmo, string $data [, bool $raw_output = false ] )
```

A función toma primeiro o algoritmo que se desexa empregar, *$algoritmo*, y despois o string que se desexa encriptar, *$data*. O algoritmo puede ser **md5, sha128, sha256**... Devolverá o contrasinal encriptado.

A avantaxe de empregar estas funcións é que podemos introducir directamente os contrasinais desde PHPMyAdmin. A desvantaxe é que NON son seguras 100%.

Librería hash de contrasinais

A extensión hash para contrasinais de PHP crea un password complexo, que se axusta aos estándares de seguridade do momento, polo que é o **método máis recomendado**.

Podemos empregar a función ***password_hash***, co contrasinal que queremos "hashear", e a extensión xa o fai directamente. Recoméndase gardar o resultado nun campo de 255 caracteres.

Empregará o algoritmo que lle indiquemos, se indicamos PASSWORD_DEFAULT empregará o algoritmo máis forte. Ver https://www.php.net/manual/es/function.password-hash

- **password_hash**

  string password_hash (string $password, integer $algoritmo [,array $options])

Por exemplo, podemos "hashear" o noso contrasinal con:

```
   $hasheado = password_hash("abc123.", PASSWORD_DEFAULT);
```

Dependendo do número ($options) o algoritmo será máis complexo e tardará máis en xerarse o hash.

Pódese considerar como o número de veces que o algoritmo "hashea" o contrasinal

Para comprobar o contrasinal gardado, agora temos que empregar a función

boolean password_verify($password, $ hash)

Devolverá TRUE se o contrasinal coincide co almacenado, e FALSE en caso contrario.

Exemplo:

```
... XA FIXEMOS A CONSULTA E TEMOS O PASSWORD GARDADO NA BD NA VARIABLE $hash:

if(password_verify($passwordTecleado, $hash)) {
     //PASSWORD CORRECTO    }
else {
   //PASSWORD INCORRECTO    }
```

Empregando esta extensión de cifrado de PHP, a nosa aplicación estará nos últimos estándares de seguridade. A extensión vaise adaptando aos cambios e a nosa aplicación será segura.

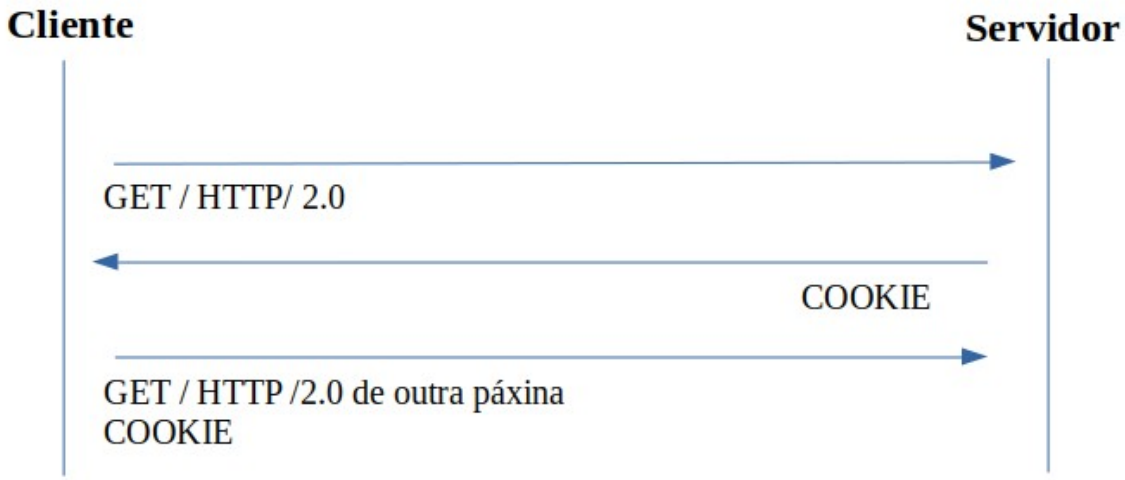
Un exemplo podería ser (se temos na nosa BD **proba** unha táboa **usuario** con campos **usuario** e **passwd**):

```
<?php

//SE NON ESTÁ AUTENTICADO PEDIMOS CREDENCIAIS
if (!isset($_SERVER['PHP_AUTH_USER'])) {
   header("WWW-Authenticate: Basic realm='Contido restrinxido'");
   header("HTTP/1.0 401 Unauthorized");
   die();
}
```

```
$host = "db-pdo";
$db = "proba";
$user = "root";
$pass = "root";
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
try {
   $conPDO = new PDO($dsn, $user, $pass);
   $conPDO->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $ex) {
   die("Erro na conexión mensaxe: " . $ex->getMessage());
}

// COMPROBAMOS SE EXISTE O USUARIO, E RECOLLEMOS O PASSWORD GARDADO NA BD
$consulta = "select passwd from usuario where usuario=:nomeTecleado";
$stmt = $conPDO->prepare($consulta);

try {
   $stmt->execute(array('nomeTecleado' => $_SERVER['PHP_AUTH_USER']));
} catch (PDOException $ex) {
   $conPDO = null;
   die("Erro recuperando os datos da BD: " . $ex->getMessage());
}
$fila=$stmt->fetch();
if($stmt->rowCount() == 1 ) //HAI UN USUARIO
   $contrasinalBD=$fila[0];

$passTecleado=$_SERVER['PHP_AUTH_PW'];

// COMPROBAMOS QUE O HASH GARDADO É COMPATIBLE CO TECLEADO.
//TEMOS QUE COMPROBAR ANTES QUE HAI ALGÚN USUARIO:
if ($stmt->rowCount() == 0 || !password_verify($passTecleado,$contrasinalBD)) {
   header("WWW-Authenticate: Basic realm='Contido restrinxido'");
   header("HTTP/1.0 401 Unauthorized");
   $stmt = null;
   $conProyecto = null;
   die();
}

$stmt = null;
$conPDO = null;
?>
<!DOCTYPE html>
<head>
<title>Autenticación BD</title>
</head>
<body>
<p>
<?php
echo "Benvido {$_SERVER['PHP_AUTH_USER']}, está vostede na área restrinxida";
?>
</p>
</body></html>
```

Cookies

Unha ***cookie*** é un fragmento de texto que se envía desde os servidores dun sitio web e que se garda no navegador web do cliente. Esta información se intercambia entre o cliente e o servidor, e permite a este último servir diferentes contidos segundo as preferencias do usuario (tamén conseguir información sobre os hábitos de navegación):



Cando o cliente pide unha páxina ao servidor este lle envía unha cookie, que o cliente enviará de volta ao servidor coas diferentes peticións seguintes.

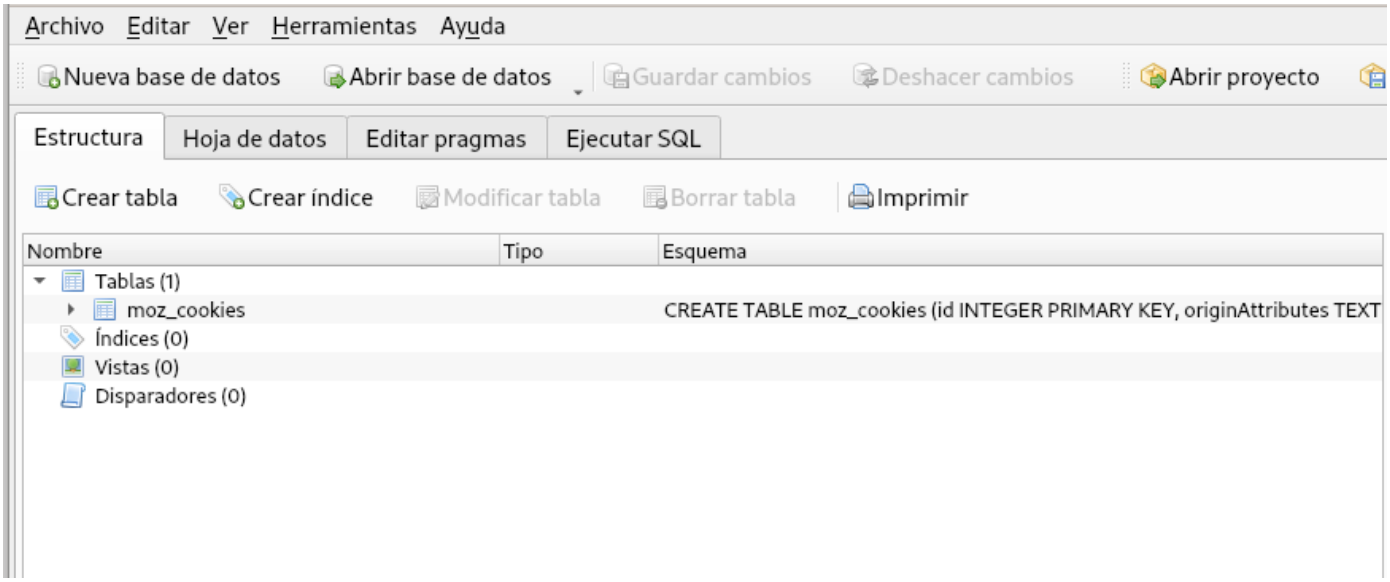
Deste modo, estes fragmentos de texto quedan rexistrados no navegador identificando a cada cliente cando cambia de páxina do servidor, e ata durante visitas de diferentes días, pois en cada petición se enviará a cookie. A idea á solucionar o problema da falta de estado na navegación web.

Onde se almacenan?

No inicio as cookies gardábanse en ficheiros de texto. Agora os navegadores almacenan nunha base de datos, sendo independentes unha das outras.

Por exemplo, Firefox almacena as cookies nunha BD sqlite dentro do perfil de usuario (na carpeta .mozilla, en Debian no Cartafol Persoal/.mozilla/firefox/9g2vffjs.default-esr/

Se abrimos ese ficheiro con algún xestor de BD de sqlite podemos acceder aos valores gardados nas cookies. A táboa ten estes campos:



Se premes na pestana "Hoja de Datos" poderás ver os valores que envía o teu navegador a diferentes servidores aos que se conectou anteriormente.

Como se envían?

As cookies envíanse empregando encabezamentos HTTP específicos, tanto nas mensaxes de petición como nas mensaxes de resposta.

Cookies e PHP

Para traballar coas cookies empregamos a función ***setcookie()*** ***(***ver https://www.php.net/manual/es/function.setcookie.php***).***

Definirá unha cookie para ser enviada nas cabeceiras do documento (terá que ser chamada antes de calquera saída de texto). Se houbo algún tipo de output anterior á chamada de setcookie(), esta devolverá false, en caso contrario devolverá true. Isto non indica que o usuario acepte a cookie.

Exemplo:

```
setcookie('Lingua','Galego');
```

Todos os argumentos agás o nome son opcionais:

- **nome**. Nome da cookie. Se creamos unha cookie soamente co nome, no cliente eliminásese a cookie que exista con ese nome. Tamén podemos substituír calquera argumento cunha cadea baleira ("").
- **value**. Valor que almacenará a cookie no cliente.

- **expire**. O argumento expire é un argumento enteiro que indica a hora en que se elimina a cookie no formato de hora que devolven as funcións UNIX time() e mktime(). Normalmente úsase time() + n segundos de duración, para especificar a duración dunha cookie. Se este valor está valeiro a cookie rematará ao pechar a sesión do navegador. Lembra que para que este argumento teña sido en conta, o ***value*** da cookie ten que existir.
- **path**. Subdirectorio onde ten valor a cookie.
- **dominio**. Dominio onde ten valor a cookie. Se pomos como dominio www.domain.com a cookie non se transmite para domain.com, mentres que se pomos domain.com a cookie transmítese tanto para domain.com como para www.domain.com
- **secure**. O argumento secure indica que a cookie só se transmite a través dunha conexión segura HTTPS.

Se queremos que a cookie sexa persistente, temos que enviar un valor para Expire. Por exemplo, 2 horas:

```
setcookie('Lingua','Galego', time( )+7200);    //os segundos de 2 horas
```

Para acceder agora aos valores enviados en PHP o servidor emprégase o array asociativo $_COOKIE, que terá o elemento creado nas páxinas creadas posteriormente á súa creación (se estamos nunha páxina teremos que recargala con header('Location:... '):

```
echo 'A lingua elexida foi ', $_COOKIE['Lingua'];
```

Podemos almacenar algún array simple nunha cookie, engadindo tantas cookies como membros teña o array, definindo o nome coa expresión de array:

```
   setcookie('user[nome]', 'Xan');
   setcookie['user[email]','xan@iessanclemente.net');
```

Para recuperar estes valores con PHP percorremos a cookie cun bucle (será un array):

```
if(!empty($_COOKIE['user'] ) ) {
   foreach ($_COOKIE['user']  as $key=>$value)
      echo "$key : $value <br>";
```

Borrar unha cookie

Para borrar unha cookie podemos empregar a mesma función setcookie() cun tempo pasado (temos que empregar os mesmos parámetros):

```
<?php
//POÑEMOS O TEMPO A MEDIA HORA ANTES:
setcookie('Lingua','Galego', time() - 1800);

?>
```

ou máis fácil:

```
<?php
//POÑEMOS O TEMPO NO PRIMEIRO SEGUNDO DA ERA LINUX (NO 1/1/1970)
setcookie('Lingua','Galego', 1);

?>
```

Con este método hai que recargar a páxina, pois ata que se recarga a páxina a cookie non se elimina, pois como podemos ver na páxina de www. php.net, para a función setcookie():

> *Las cookies no se volverán visibles hasta la próxima carga de la página en la que debieran serlo. Para probar si se ha creado correctamente una cookie, se debe buscar la cookie en alguna página cargada posteriormente y antes que la cookie expire.*

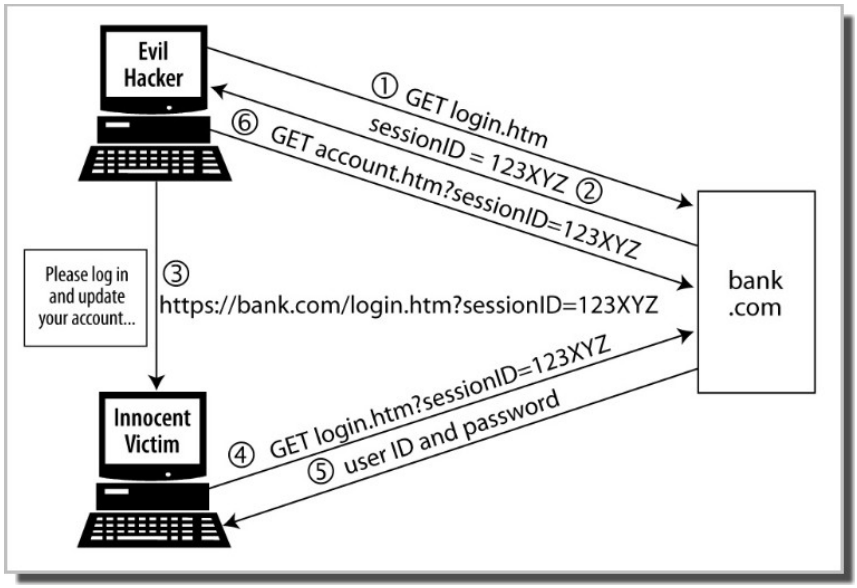
Algúns apuntes de seguridade en PHP

Imos ver neste apartado algúns apuntes de seguridade en PHP. Este campo é tremendamente extenso, polo que este apartado pretende ser só unha introdución á seguridade en PHP.

Session Fixation (Fixación da sesións)

Que é session fixation?

Unha fixación de sesión é un ataque que permite a un atacante secuestrar ou fixar unha session-ID válida dun usuario, para empregala suplantando a identidade dun usuario. Na figura o hacker conéctase ao banco, e logo envía un enlace á víctima fixando o identificador da sesión. Cando a víctima se conecta ao banco, o seu SID está fixado, de modo que o hacker pode conectarse a posteriori con ese identificador de sesión (no exemplo 123XYZ).



Evitar ataque por Session Fixation

Nas versións modernas de PHP a propagación das ID's de sesión só se realixa empregando cookies (non se envía coa URL). Isto está marcado pola variable ***session.use_only_cookies*** (Que por defecto está a 1). Pero é conveniente ter certas prácticas para evitar este ataque

Rexenerar o SID nas sesións novas

Para que cada sesión nova colla un valor diferentes para session-ID, debemos empregar a función ***session_regenerate_id(true)*** para sesións novas. Para saber se a sesión é nova, crearemos unha variable a primeira vez que se crea a sesión. Se non existe esa marca, cambiaremos o ID da sesión por un novo. Así, faremos:

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

```
if(!isset($_SESSION[‘marcadecontrol’]){
   session_regenerate_id(true);   //borrarmos o ficheiro da ID da sesión anterior
   $_SESSION[‘marcadecontrol’]= true;
}
?>
```

Cambiar o id cada vez que o usuario se loguee

Outra posibilidade é cambiar o id da sesión sempre que o noso usuario se loguee

```
<?php
if ($usuario_logueado === true)
{
     session_regenerate_id(true);
     $_SESSION['logueado'] = true;
}
?>
```

Para saber máis: https://en.wikipedia.org/wiki/Session_fixation

Introdución a XSS (Cross Site Scripting)

Que é?

Un ataque **XSS** (***Cross Site Scripting***) consiste en introducir código ***HTML*** ou ***Javascript*** nas caixas de texto dos formularios, e se non se controlan os datos que se introducen, pódese facer que o código se execute na web.

Por exemplo, se un usuario escribe o seguinte nunha caixa de texto, e non se controlan os datos introducidos, a web se redirixe a google.com:

```
<script>window.location = "http://www.google.com";</script>
```

Outro exemplo tampouco malicioso sería escribir un alert( ) na nosa caixa de texto:

```
<script>alert("Como estás??”);</script>
```

Para saber máis: https://en.wikipedia.org/wiki/Cross-site_scripting

Evitando ataques XSS

PHP dispón de algunhas funcións para previr este tipo de ataques XSS: ***htmlspecialchars( ), htmlentities( )***, e ***strip_tags( ).***

Normalmente ***htmlspecialchars( )*** é suficiente para filtrar a saída do contido que se amosa no navegador. Se empregamos algún tipo de codificación distintas de UTF-8, empregaremos ***htmlentities( )*** para ese filtrado.

```
echo htmlspecialchars('<script>alert("Como estás??”);</script>');
// resultado: &lt;script&gt;alert(&quot;Como est&aacutes??&quot;);&lt;/script&gt;

echo htmlentities('<script>alert("Como estás??");</script>');
// resultado: &lt;script&gt;alert(&quot;Como est&aacutes??&quot;);&lt;/script&gt;
```

Se queremos protexer o contido que se almacena nunha táboa, podemos facer limpeza das etiquetas html antes de almacenalas empregando a función strip-tags

```
$texto="<script>alert('Como estás??')<script>";
$txtSenEtiquetas= strip_tags($texto);   // alert(‘Como estás??’)
```

Se queremos un control maior podemos empregar a clase ***InputFilter***, que veremos máis adiante.