IES NTRA. SRA. DE LOS REMEDIOS DE UBRIQUE

Tema 11 – Seguridad

2º DAW – Desarrollo Web Entorno Servidor

Profesor Juan Carlos Moreno

CURSO 2024/2025

Tabla de Contenido

11 Seguridad

11.1 Introducción

En la actualidad el crecimiento de internet ha impactado directamente en la seguridad de la información manejada cotidianamente. Sitios de comercio electrónico, servicios, bancos e incluso redes sociales contienen información sensible que en la mayoría de los casos resulta ser muy importante.

Se puede decir que uno de los puntos más críticos de la seguridad en Internet son las herramientas que interactúan de forma directa con los usuarios, en este caso los servidores web. Es común escuchar sobre fallos en los sistemas de protección de los servidores más frecuentemente utilizados, por ejemplo Apache, NGINX, IIS, etc. (Build With, 2016) O en los lenguajes de programación en que son escritas las aplicaciones. Sin embargo, la mayoría de los problemas detectados en servicios web no son provocados por fallos de ninguna de estas partes, si no que los problemas se generan por malas prácticas de parte de los programadores.

Debemos entender que programar aplicaciones web seguras no es una tarea fácil, ya que requiere por parte del programador, no sólo cumplir con el objetivo funcional básico de la aplicación, sino una concepción general de los riesgos que puede correr la información procesada por el sistema.

Gran parte de los problemas de seguridad en las aplicaciones web son causados por la falta de seguimiento en dos rubros muy importantes de los que depende cualquier aplicación, las entradas y salidas del sistema

Además de verificar estos 2 rubros, es importante considerar la exposición accidental de datos que pueden ser empleados en un posible ataque sobre el sistema. Los mensajes de error enviados por el servidor, que suelen ser de gran utilidad durante el proceso de desarrollo de la aplicación, pueden ser empleados maliciosamente cuando siguen apareciendo en un entorno de producción, por lo que es necesario deshabilitar todos estos mensajes y editar algunos otros (como los que se envían cuando el servidor no encuentra algún archivo en particular) los cuales también pueden ser utilizados por los atacantes para obtener información sobre nuestro sistema.

Veamos a continuación algunos capítulos relacionados con el desarrollo de aplicaciones web más seguras.

11.2 Encriptación y Contraseñas

Desde el principio PHP ha sido un lenguaje de programación para la construcción de sitios web. Esa idea permanece en el núcleo del lenguaje, y por eso es tan popular para la construcción de aplicaciones web. Cuando se creó en los años 90, el término aplicación web no existía aún, por lo que la protección de contraseñas para cuentas de usuarios no era algo en lo que estuviera centrado.

Han pasado muchos años desde entonces y actualmente es impensable una aplicación web que no proteja las cuentas de los usuarios con contraseñas. Es fundamental para cualquier programador hacer que estas contraseñas tengan una encriptación segura y eficiente. A partir de PHP 5.5 se añadió una nueva librería llamada Hash de contraseñas para la encriptación de contraseñas, con funciones que facilitan la tarea y utilizan los últimos métodos más eficaces.

11.2.1 La importancia de los hashes seguros

Siempre hay que guardar las contraseñas encriptadas mediante un algoritmo de encriptación como el algoritmo hashing para hacer imposible a alguien que acceda a una base de datos conseguir averiguar la contraseña. Esto no es sólo para proteger a los usuarios frente a algún atacante sino también frente a los propios empleados de la aplicación.

Mucha gente utiliza las mismas contraseñas para muchas aplicaciones web. Si alguien accede a la dirección de email y contraseña de un usuario, probablemente pueda hacerlo en muchas otras aplicaciones.

Los hashes no se crean iguales, se emplean algoritmos muy distintos para crear un hash. Los dos más usados en el pasado son MD5 y SHA-1. Los ordenadores de hoy en día pueden crackear fácilmente estos algoritmos. Dependiendo de la complejidad y longitud de la contraseña, se puede crackear en menos de una hora con los dos algoritmos nombrados (los ratios son 3650 millones de cálculos por segundo con MD5 y 1360 millones por segundo con SHA-1).

Por eso es importante usar algoritmos complejos. Si el hash es más largo reduce el riesgo de colisiones entre contraseñas (dos frases generando el mismo hash), pero también conviene que la aplicación se tome el tiempo necesario para generar el hash. Esto es porque el usuario apenas notará un segundo o dos más de tiempo de carga al logearse, pero se consigue que crackearlo tome muchisimo más tiempo, en case de que sea posible.

11.2.2 Métodos de encriptación

Primero veamos la funciones básicas de hashing para PHP:

md5

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

Calcula un hash con el algoritmo md5. Si se establece _$rawoutput como true se devolverá en raw binario con una longitud de 16. De normal devuelve un hash de 32 caracteres hexadecimal.

sha1

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

Calcula un hash con el algoritmo sha1. Si se establece _$rawoutput como true se devolverá en raw binario con una longitud de 20. De normal devuelve un hash de 40 caracteres hexadecimal.

hash

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

La función toma primero el algoritmo que se desea emplear, $algo, y después el string que se desea encriptar, $data. El algoritmo puede ser md5, sha128, sha256...

Anteriormente, el siguiente código era un ejemplo de una buena protección de contraseñas:

```
class Password {
    const SALT = 'EstoEsUnSalt';
    public static function hash($password) {
        return hash('sha512', self::SALT . $password);
    }
    public static function verify($password, $hash) {
        return ($hash == self::hash($password));
    }
}
// Crear la contraseña:
$hash = Password::hash('micontraseña');
// Comprobar la contraseña introducida
if (Password::verify('micontraseña', $hash)) {
    echo 'Contraseña correcta!\n';
} else {
    echo "Contraseña incorrecta!\n";
}
```

Durante mucho tiempo esto ha sido la mejor forma de protegerse, mejor que usar md5. Se usa un algoritmo mucho más complejo como el sha512, y fuerza a todas las contraseñas a usar un salt, pero tiene algunas carencias:

- Se utiliza un salt, pero todas las contraseñas utilizan el mismo, por lo que si alguien consigue averiguar una contraseña, o el acceso al código fuente donde puede mirar el hash, se puede hacer una Rainbow Table añadiendo el salt descubierto. La solución es crear un salt aleatorio para cada contraseña que se crea, y guardar el salt con la contraseña de forma que después se pueda recuperar.
- Se utiliza sha512, un complejo algoritmo que viene con PHP. Sin embargo también puede ser crackeado a un ratio de 46 millones de cálculos por segundo. Aunque es más lento de crackear que md5 y sha1, todavía no es un nivel de seguridad estable. La solución es utilizar algoritmos que son todavía más complejos y emplearlos varias veces. Por ejemplo emplear un algoritmo sha512 10 veces consecutivamente reduciría el intento de hackeo considerablemente.

Las dos soluciones ya vienen por defecto con la librería Hash de contraseñas de PHP.

11.2.3 La librería Hash de contraseñas de PHP

La extensión Hash de contraseñas crea un password muy complejo, incluyendo la generación de salts aleatorios. En forma más simple se utiliza la función _passwordhash(), con la contraseña que quieres "hashear", y la extensión lo hace directamente. Es necesario facilitar también el algoritmo que se desea emplear. La mejor opción de momento es especificar PASSWORD_DEFAULT (se actualiza siempre que se añada un algoritmo nuevo más fuerte), aunque también es posible PASSWORD_BCRYPT.

password_hash

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

Es compatible con crypt() por lo que los hash de contraseñas creados con crypt() se pueden usar con _passwordhash(). Las opciones que se admiten son salt (para proporcionarlo manualmente, pero esta opción ya está obsoleta en PHP 7 por lo que no conviene usarla) y cost, que denota el coste del algoritmo a usar (el valor predeterminado es 10).

```
$hash = password_hash('micontraseña', PASSWORD_DEFAULT, [15]);
```

El coste indica cuánto de complejo debe ser el algoritmo y por lo tanto cuánto tardará en generarse el hash. El número se puede considerar como el número de veces que el algoritmo hashea la contraseña.

Para poder verificar los passwords, deberíamos saber el salt que se ha creado. Si se usa passwordhash() otra vez y se compara con el anterior, se puede ver que son distintos. Cada vez que se llama a la función, se genera un nuevo hash, por lo que la extensión facilita una segunda función: password_verify(). Llamando a esta función y pasando la contraseña proporcionada por el usuario, la función devolverá true si coincide con la almacenada:

```
if(password_verify($password, $hash)){
    // Password correcto!
}
```

Ahora la clase que habíamos puesto al principio se puede refactorizar por una mucho más segura:

```
class Password {
    public static function hash($password) {
        return password_hash($password, PASSWORD_DEFAULT, ['cost' => 15]);
    }
    public static function verify($password, $hash) {
        return password_verify($password, $hash);
    }
}
```

11.2.4 Cambios en los métodos de encriptación

Usando la extensión de encriptación de PHP, tu aplicación estará con los últimos estándares en seguridad, aunque hace algunos años de decía que SHA-1 era lo mejor. Esto significa que cada vez se van actualizando los logaritmos y por tanto la extensión se irá adaptando.

¿Y qué ocurre con las contraseñas antiguas? Para eso está la función _password_needsrehash(), que detecta si una contraseña almacenada no cumple con las necesidades de seguridad de la aplicación. La razón puede ser que hayas aumentado la complejidad con cost, o que PHP haya actualizado el algoritmo. Por esta razón se ha de elegir PASSWORD_DEFAULT, siempre se estará protegido con la opción más segura disponible.

Cuando un usuario se logea, ahora tendríamos una nueva tarea, llamar a _password_needsrehash(), que toma parámetros similares a _passwordhash(). Lo que hace la función _password_needsrehash() es decirte si el password necesita un rehash. Depende de ti cuándo generar un nuevo hash de contraseña y guardarlo, porque la extensión Hash desconoce cómo deseas hacerlo.

El siguiente es un ejemplo de una clase Usuario simulada para ver el funcionamiento de la extensión Hash, y sirve de orientación a cómo podria realizarse:

```
class User {
    // Opciones de contraseña:
    const HASH = PASSWORD_DEFAULT;
    const COST = 14;
    // Almacenamiento de datos del usuario:
    public $data;
    // Constructor simulado:
    public function __construct() {
        //  Leer los datos de la base de datos almacenados en $data, como
        //  $data->passwordHash  o  $data->username
    }
    // Funcionalidad de guardar los datos simulada:
    public function save() {
        // Guardar los datos de $data en la base de datos
    }
```

Ya hemos construido la base de la clase user, ahora vamos a ver el cambio de contraseña y el login:

```
// Permite el cambio de contraseña:
    public function setPassword($password) {
        $this->data->passwordHash = password_hash($password, self::HASH,
['cost' => self::COST]);
    }
    // Logear un usuario:
    public function login($password) {
        // Primero comprobamos si se ha empleado una contraseña correcta:
        echo "Login: ", $this->data->passwordHash, "\n";
        if (password_verify($password, $this->data->passwordHash)) {
            // Exito, ahora se comprueba si la contraseña necesita un rehash:
            if (password_needs_rehash($this->data->passwordHash, self::HASH,
['cost' => self::COST])) {
                // Tenemos que hacer rehash en la contraseña y guardarla.
Simplemente se llama a setPassword():
                $this->setPassword($password);
                $this->save();
            }
            return true; // O hacer lo necesario para indicar que el usuario
se ha logeado.
        }
        return false;
    }
}
```

11.3 Filtrado de Datos de Entrada

11.3.1 Validación con Expresiones Regulares

En cualquier formulario de cualquier aplicación siempre hay que validar los datos que introducen los usuarios. Una validación simple puede consistir en asegurarse de que se rellena un campo, se puede hacer fácilmente con empty():

```
<form action="index.php" method="post">
Nombre: <input type="text" name="nombre"><br>
Email: <input type="text" name="email"><br>
Enviar <input type="submit" name="enviar">
</form>
```

```
<?php
if(isset($_POST['enviar'])) {
    if(empty($_POST["nombre"])){
        echo "El nombre es requerido";
    } else {
    $nombre = $_POST["nombre"];
}
    // Más código...
}
```

Si el nombre no se rellena, el formulario devolverá "El nombre es requerido".

Para validar campos de una forma más precisa (y compleja) se utilizan expresiones regulares. Para ello PHP dispone de la función preg_match() que exige dos parámetros obligatorios, la expresión regular y el string que tiene que comprobar. El siguiente es un ejemplo que asegura que el nombre sólo sean letras y el email tenga un formato adecuado de emails (empleando el mismo formulario html anterior):

```
<?php
$errores = array();
if(isset($_POST['enviar'])) {
    // Requerimos el nombre:
    if (empty($_POST["nombre"])) {
        $errores[] = "El nombre es requerido <br>";
    } else {
        $nombre = $_POST["nombre"];
        // Queremos que el nombre de usuario sólo tenga letras
        if (!preg_match("/^[a-zA-Z]+/", $nombre)) {
            $errores[] = "Sólo se permiten letras como nombre de usuario
<br>";
        }
    }
    // Requerimos el email también:
    if (empty($_POST["email"])) {
        $errores[] = "El email es requerido <br>";
    } else {
        $email = $_POST['email'];
        // Queremos que el email tenga un formato adecuado
        if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/", $email)) {
            $errores[] = "Formato de email incorrecto";
        }
    }
    if(empty($errores)) {
        echo "Nombre: $nombre <br>";
        echo "Email: $email <br>";
    } else {
        var_dump($errores);
    }
}
```

Con las expresiones regulares se consigue un nivel de precisión muy alto. De todas formas, PHP tiene una extensión de filtrado con la que se pueden realizar filtros más rápidos para la validación y saneamiento de los datos. Si es posible evitar el uso de la expresión regular sin afectar al nivel de validez de los datos, mejor.

11.3.2 Codificación de Caracteres

El data sanitization o la sanitización de datos consiste en modificar los inputs de entrada de forma que se eliminen algunos caracteres indeseables y se normalicen los datos para emplearlos o guardarlos de forma segura. Si por ejemplo tenemos un sistema de búsqueda en el que mostramos lo que se ha buscado al mostrar los resultados:

```
if(isset($_GET['query'])){
    echo '<p>Los resultados de tu búsqueda ', htmlspecialchars($_GET['query'],
ENT_QUOTES), ' son: </p>'; // Código para mostrar los resultados de búsqueda }
```

Con la función htmlspecialchars() reducimos el riesgo, pero siguen siendo posibles los ataques XSS. Con ENT_QUOTES se consigue que todas las comillas simples ' y dobles " se filtren, pero se pueden realizar ataques sin utilizar comillas. Tenemos que asegurarnos de utilizar la misma codificación de caracteres en la que se define el documento HTML para reducir más los riesgos:

```
// Definimos la cofificación en el HTTP header:
header('Content-Type: text/html; charset=UTF-8');
// Definimos la codificación empleada al convertir caracteres:
echo htmlspecialchars($_GET['query'], ENT_QUOTES, 'UTF-8');
```

11.3.3 Extensión de Filtrado de PHP

La extensión de filtrado de PHP permite validar o sanitizar los datos, especialmente cuando vienen de fuentes externas (como usuarios). La extensión se divide entre filtros de validación, para comprobar que los datos cumplen ciertos requisitos, y en filtros de saneamiento, que limpiará los datos de forma que se eliminen los caracteres no deseados.

Las banderas u opciones de filtrado sirven para modificar el comportamiento en el saneamiento o la validación según las necesidades. Por ejemplo, si se pasa FILTER_FLAG_PATH_REQUIRED en una url, se exige que se precise una ruta concreta: /seccion en http://www.ejemplo.com/seccion.

Funciones de filtrado

filter_list

```
array filter_list (void)
```

Devuelve un array con todos los filtros soportados.

filter_var

```
mixed filter_var (mixed $variable [, int $filter = FILTER_DEFAULT [, mixed
$options ]] )
```

Filtra una variable $variable con el filtro indicado en $filter. El filtro por defecto **FILTER_DEFAULT** equivale a **FILTER_UNSAFE_RAW**, y no filtra realmente nada.

```
// EJEMPLO 1
$var = filter_var('hola', FILTER_VALIDATE_BOOLEAN); // Valor: FALSE
// EJEMPLO 2
$opciones = array(
    'options' => array(
        'default' => 3, // Valor a devolver si falla el filtro
        'min_range' => 0
    ),
    'flags' => FILTER_FLAG_ALLOW_OCTAL,
```

```
);
$var = filter_var('12345', FILTER_VALIDATE_INT, $opciones); // Valor: 12345
```

Si se quiere validar el integer 0 con FILTER_VALIDATE_INT devuelve que "Integer is not valid". Para solucionar esto:

```
$int = 0;
if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int,
FILTER_VALIDATE_INT) === false) {
    echo("El integer es válido");
} else {
    echo("El integer NO es válido");
}
```

Para asegurarnos de que un string es un email, con la extensión de filtrado de PHP podemos sanitizar y validar un email:

```
$email = "nombre@ejemplo.com";
// Primero eliminamos cualquier carácter que pueda dar problemas
$email = filter_var($email, FILTER_SANITIZE_EMAIL);
// Luego validamos el email
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
    echo("$email es una dirección de email válida");
} else {
    echo("$email NO es una dirección de email válida");
}
```

También podemos sanitizar y validar una URL:

```
$url = "http://www.google.com";
// Eliminar cualquier carácter que pueda dar problemas
$url = filter_var($url, FILTER_SANITIZE_URL);
// Luego validamos la URL
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {
    echo("$url es una URL válida");
} else {
    echo("$url no es una URL válida");
}
```

filter_var_array

```
mixed filter_var_array (array $data [, mixed $definition [, bool $add_empty =
true ]] )
```

Devuelve los datos del array $data y opcionalmente los filtra. $definition puede ser otro array que define filtros específicos para cada uno de los elementos de $data, o simplemente una constante de filtro como FILTER_VALIDATE_URL, por ejemplo.

```
$inputs = ["<a>perro</a>", "gato"];
$animales = filter_var_array($inputs, FILTER_SANITIZE_STRING);
var_dump($inputs);
/*
 Devuelve
array (size=2)
  0 => string '<a>perro</a>' (length=38)
  1 => string 'gato' (length=4)
 */
var_dump($animales);
/*
 Devuelve
array (size=2)
```

```
  0 => string 'perro' (length=5)
  1 => string 'gato' (length=4)
 */
```

filter_input

```
mixed filter_input (int $type, string $variable_name [, int $filter =
FILTER_DEFAULT [, mixed $options ]] )
```

Toma una variable externa de un determinado tipo y opcionalmente la filtra. Estas variables pueden venir de $_GET, $_POST, $_COOKIE, $_SERVER, $_ENV, por lo que type deberá ser uno de los siguientes valores: INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER o INPUT_ENV. Los filtros y opciones funcionan de la misma forma que en las funciones anteriores.

```
$busqueda = filter_input(INPUT_GET, 'buscar', FILTER_SANITIZE_SPECIAL_CHARS);
$url = filter_input(INPUT_GET, 'buscar', FILTER_SANITIZE_ENCODED);
echo "Has buscado $busqueda.\n";
echo "Buscar otra vez";
// Ejemplo con "5f F$ "·$ %!"
// Busqueda: 5f F$ "·$ %!
// URL: 5f%20F%24%20"·%24%20%25%21
```

filter_input_array

```
mixed filter_input_array (int $type [, mixed $definition [, bool $add_empty =
true ]] )
```

Obtiene variables externas de un determinado tipo (GET, POST...) y opcionalmente las filtra. Es igual que _filter_input_ pero permite facilitar un array $definition para filtrar todos los valores de un array externo de vez.

```
<form action="index.php" method="post">
    Nombre de usuario: <input type="text" name="usuario"><br>
    Email: <input type="text" name="email"><br>
    Enviar <input type="submit" name="enviar">
</form>
<?php
if(isset($_POST['enviar'])) {
$usuario = filter_input_array(INPUT_POST, FILTER_SANITIZE_ENCODED);
var_dump($usuario);
}
/* Si pasamos:
  usuario = diego<a href>
  email = diego@ejemplo.com'$"
  Devuelve:
array (size=3)
  'usuario' => string 'diego%3Ca%20href%3E' (length=19)
  'email' => string 'diego%40ejemplo.com%27%24%22' (length=28)
  'enviar' => string 'Submit' (length=6)
*/
```

filter_has_var

```
bool filter_has_var ( int $type, string $variable_name)
```

Comprueba si existe una variable en un tipo concreto (GET, POST, COOKIE...)

```
$_GET['hey'] = "hola";
echo filter_has_var(INPUT_GET, 'hey') ? 'Si' : 'No';
// No devolverá 'Sí' a no ser que accedas desde URL con
```

```
// el parámetro hey y el valor hola:
// http://ejemplo.com/index.php?hey=hola
```

11.3.4 Constanstes de Filtrado

Las constantes de filtrado son:

Constante	Validación
FILTER_VALIDATE_BOOLEAN	Valida un booleano
FILTER_VALIDATE_EMAIL	Valida una dirección de email
FILTER_VALIDATE_FLOAT	Valida un float
FILTER_VALIDATE_INT	Valida un integer
FILTER_VALIDATE_IP	Valida una dirección IP
FILTER_VALIDATE_REGEXP	Valida una expresión regular
FILTER_VALIDATE_URL	Valida una URL
FILTER_SANITIZE_EMAIL	Elimina caracteres peligrosos de una direcció
FILTER_SANITIZE_ENCODED	Elimina caracteres especiales
FILTER_SANITIZE_MAGIC_QUOTES	Hace lo mismo que la función addslashes()
FILTER_SANITIZE_NUMBER_FLOAT	Elimina todo menos dígitos, +, - y opcionalm
FILTER_SANITIZE_NUMBER_INT	Elimina todo menos dígitos
FILTER_SANITIZE_SPECIAL_CHARS	Elimina caracteres especiales
FILTER_SANITIZE_STRING	Elimina etiquetas y caracteres especiales de u
FILTER_SANITIZE_STRIPPED	Alias de FILTER_SANITIZE_STRING
FILTER_SANITIZE_URL	Elimina caracteres peligrosos de una URL
FILTER_UNSAFE_RAW	No hace nada, igual que FILTER_DEFAULT
FILTER_CALLBACK	Llama a una función callback definida por el u

11.4 Formularios

11.4.1 Validación Completa Formulario

La validación de formularios es algo fundamental ya que previene posibles ataques de intrusos, además de asegurar que los datos que se reciben son realmente del tipo deseado.

- Existen dos formas de validación de formularios: en el lado del cliente y en el lado del servidor. En el lado del cliente la validación suele ser mediante JavaScript, es más rápida y evita enviar más trabajo al servidor.
- En el lado del servidor se emplea PHP para verificar que se envían valores correctos, es más seguro pero es más lento y da un poco de trabajo al servidor. En este capítulo vamos a ver algún ejemplo de esta segunda forma.

El siguiente es un sencillo formulario con los siguiente datos: Nombre, Contraseña, Educacion, Nacionalidad, Idiomas, Email y Sitio web:

```
<h2>Formulario:</h2>
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
method="post">
    Nombre:
<input type="text" name="nombre" maxlength="50"><br>
    Contraseña:
<input type="password" name="password"><br>
    Educacion:
<select name="educacion">
<option value="sin-estudios">Sin estudios</option>
<option value="educacion-obligatoria" selected="selected">Educación
Obligatoria</option>
<option value="formacion-profesional">Formación profesional</option>
<option value="universidad">Universidad</option>
</select><br>
    Nacionalidad:
<input type="radio" name="nacionalidad" value="hispana">Hispana</input>
<input type="radio" name="nacionalidad" value="otra">Otra</input><br>
    Idiomas:
<input type="checkbox" name="idiomas[]" value="español"
checked="checked">Español</input>
<input type="checkbox" name="idiomas[]" value="inglés">Inglés</input>
<input type="checkbox" name="idiomas[]" value="francés">Francés</input>
<input type="checkbox" name="idiomas[]" value="aleman">Alemán</input><br>
    Email:
<input type="text" name="email"><br>
    Sitio Web:
<input type="text" name="sitioweb"><br>
    // Botón de enviar
<input type="submit" name="submit" value="Enviar">
</form>
</body>
</html>
```

Vamos a ver los aspectos más característicos:

- El atributo maxlength limita el número de caracteres para una caja de texto.
- Contraseña es del tipo password, lo que permite ocultar los caracteres con * cuando se escribe.

- Educación es un elemento select, que permite seleccionar options en una lista. El texto en el atributo value es el que se enviará a través del formulario. El atributo selected permite mostrar el seleccionado por defecto.
- Nacionalidad es del tipo radio, los elementos radio del mismo grupo han de tener un único atributo name y sólo se puede seleccionar uno.
- Los idiomas están en un elemento checkbox, que permite enviar varios datos en forma de array si se indica en el atributo name con corchetes. Se puede añadir un atributo checked como seleccionado por defecto.
- En action se ha puesto _<?php echo htmlespecialchars($_SERVER["PHPSELF"]); ?>. $_SERVER["PHP_SELF"] es una superglobal que devuelve el nombre del archivo en el que se encuentra el formulario, lo que hace que los datos se envíen al mismo archivo, en lugar de llevarlos a otro archivo para tratarlos. Si se emplea esta forma de indicar el archivo para action, es necesario usar la función htmlespecialchars(), que convierte caracteres especiales en entidades HTML previniendo posibles ataques Cross-site Scripting.

En este ejemplo todas las acciones de validar, enviar y recibir datos se hacen en el mismo archivo form.php.

Primero vamos a crear un filtrado común para todos los campos con la función filtrado():

```
function filtrado($datos){
    $datos = trim($datos); // Elimina espacios antes y después de los datos
    $datos = stripslashes($datos); // Elimina backslashes \
    $datos = htmlspecialchars($datos); // Traduce caracteres especiales en
entidades HTML
    return $datos;
}
```

Esta función se aplicaría a cada campo a la hora de recibir los datos:

```
if(isset($_POST["submit"]) && $_SERVER["REQUEST_METHOD"] == "POST"){
    $nombre = filtrado($_POST["nombre"]);
    $password = filtrado($_POST["password"]);
    $educacion = filtrado($_POST["educacion"]);
    $nacionalidad = filtrado($_POST["nacionalidad"]);
    // Utilizamos implode para pasar el array a string
    $idiomas = filtrado(implode(", ", $_POST["idiomas"]));
    $email = filtrado($_POST["email"]);
    $sitioweb = filtrado($_POST["sitioweb"]);
}
```

Para aceptar los datos se ha puesto isset($POST["submit"]), que crea la key submit cuando se hace click en el botón, y _$_SERVER["REQUESTMETHOD"] == "post", que especifica que el método request ha de ser POST. Se puede usar una de las dos formas o las dos a la vez.

Para mostrar los datos:

```
<?php if(isset($_POST["submit"])): ?>
<h2>Mostrar datos enviados</h2>
Nombre : <?php isset($nombre) ? print $nombre : ""; ?><br>
Contraseña : <?php isset($password) ? print $password : ""; ?><br>
Educación : <?php isset($educacion) ? print $educacion : ""; ?><br>
Nacionalidad : <?php isset($nacionalidad) ? print $nacionalidad : "";?><br>
Idiomas : <?php isset($idiomas) ? print $idiomas : ""; ?><br>
```

```
Email : <?php isset($email) ? print $email : ""; ?><br>
Sitio web : <?php isset($sitioweb) ? print $sitioweb : ""; ?><br>
<?php endif; ?>
```

Todos los campos anteriores son opcionales. La validación más básica consiste en exigir que se rellene un campo, y comprobaremos también que el email tiene un formato de email, que el sitio web es una URL correcta y que password tiene un mínimo de 5 caracteres. Las validaciones de formatos pueden realizarse mediante expresiones regulares o con filtros. Siempre que sea posible, es mejor utilizar los filtros de PHP.

```
if(isset($_POST["submit"]) && $_SERVER["REQUEST_METHOD"] == "POST"){
    // El nombre y contraseña son campos obligatorios
    if(empty($_POST["nombre"])){
        $errores[] = "El nombre es requerido";
    }
    if(empty($_POST["password"]) || strlen($_POST["password"]) < 5){
        $errores[] = "La contraseña es requerida y ha de ser mayor a 5
caracteres";
    }
    // El email es obligatorio y ha de tener formato adecuado
    if(!filter_var($_POST["email"], FILTER_VALIDATE_EMAIL) ||
empty($_POST["email"])){
        $errores[] = "No se ha indicado email o el formato no es correcto";
    }
    // El sitio web es obligatorio y ha de tener formato adecuado
    if(!filter_var($_POST["sitioweb"], FILTER_VALIDATE_URL) ||
empty($_POST["sitioweb"])){
        $errores[] = "No se ha indicado sitio web o el formato no es
correcto";
    }
    // Si el array $errores está vacío, se aceptan los datos y se asignan a
variables
    if(empty($errores)) {
        $nombre = filtrado($_POST["nombre"]);
        $password = filtrado($_POST["password"]);
        $educacion = filtrado($_POST["educacion"]);
        $nacionalidad = filtrado($_POST["nacionalidad"]);
        // Utilizamos implode para pasar el array a string
        $idiomas = filtrado(implode(", ", $_POST["idiomas"]));
        $email = filtrado($_POST["email"]);
        $sitioweb = filtrado($_POST["sitioweb"]);
    }
}
```

El array de errores $errores guarda cada uno de los errores que se registran para luego mostrarlos. Si el array tiene algún elemento, el formulario no será aceptado. Para mostrarlos podemos hacerlo así:

```
<ul>
<?php if(isset($errores)){
    foreach ($errores as $error){
        echo "<li> $error </li>";
    }
}
?>
</ul>
```

11.4.2 Subida de Archivos

Lo más laborioso a la hora de hacer un formulario que permita la subida de archivos es manejarlos y organizarlos una vez están en el servidor. Inicialmente los archivos se suben a un directorio temporal y luego se relocalizan.

Configuración en PHP para la subida de archivos:

- Debe estar On la variable file_uploads en php.ini.
- En php.ini también se encuentra la variable upload_tmp_dir, que indica el directorio donde se dirigirán los archivos cuando se suban. El tamaño máximo viene indicado en el mismo archivo de configuración por upload_max_filesize. Para ver estos valores se puede utilizar ini_get('upload_tmp_dir') e ini_get('upload_max_filesize'). upload_tmp_dir sólo es manipulable desde el php.ini o httpf.conf, upload_max_filesize puede también desde .htaccess o un user.ini (tienen modo PHP_INI_SYSTEM y PHP_INI_PERDIR respectivamente).
- Es necesario que los permisos tanto del directorio temporal como del directorio final sean de escritura.
- Para que un formulario tenga la capacidad de aceptar archivos se añade el atributo enctype="multipart/form-data" al elemento form.

De nuevo en este ejemplo pondremos todo el contenido en un archivo form2.php:

```
<h2>Formulario subida de archivos</h2>
<html>
<body>
<form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
method="POST" enctype="multipart/form-data">
<input type="hidden" name="MAX_FILE_SIZE" value="<?php echo $max_file_size;
?>" />
<input type="file" name="imagen" />
<input type="submit" name="submit" />
</form>
</body>
</html>
```

Se incluye un input del tipo hidden que limita el tamaño máximo del archivo que se puede subir (en bytes).

La información sobre un archivo subido la proporciona el array multidimensional $_FILES. Este array se crea con el key que se indique en el input del formulario, en este caso "imagen":

```
$_FILES["imagen"]["name"]. Guarda el nombre original del archivo del cliente.
$_FILES["imagen"]["type"]. Guarda el MIME type del archivo.
$_FILES["imagen"]["size"]. Guarda el tamaño del archivo en bytes.
$_FILES["imagen"]["tmp_name"]. Guarda el nombre del archivo temporal.
$_FILES["imagen"]["error"]. Guarda cualquier código de error que pueda
provocar la subida del archivo.
```

La función move_uploaded_file() mueve un archivo subido del directorio temporal al directorio que se indique.

```
$directorioSubida = "uploads/";
$max_file_size = "51200";
$extensionesValidas = array("jpg", "png", "gif");
```

```
if(isset($_POST["submit"]) && isset($_FILES['imagen'])){
    $errores = array();
    $nombreArchivo = $_FILES['imagen']['name'];
    $filesize = $_FILES['imagen']['size'];
    $directorioTemp = $_FILES['imagen']['tmp_name'];
    $tipoArchivo = $_FILES['imagen']['type'];
    $arrayArchivo = pathinfo($nombreArchivo);
    $extension = $arrayArchivo['extension'];
    // Comprobamos la extensión del archivo
    if(!in_array($extension, $extensionesValidas)){
        $errores[] = "La extensión del archivo no es válida o no se ha subido
ningún archivo";
    }
    // Comprobamos el tamaño del archivo
    if($filesize > $max_file_size){
        $errores[] = "La imagen debe de tener un tamaño inferior a 50 kb";
    }
    // Comprobamos y renombramos el nombre del archivo
    $nombreArchivo = $arrayArchivo['filename'];
    $nombreArchivo = preg_replace("/[^A-Z0-9._-]/i", "_", $nombreArchivo);
    $nombreArchivo = $nombreArchivo . rand(1, 100);
    // Desplazamos el archivo si no hay errores
    if(empty($errores)){
        $nombreCompleto = $directorioSubida.$nombreArchivo.".".$extension;
        move_uploaded_file($directorioTemp, $nombreCompleto);
        print "El archivo se ha subido correctamente";
    }
}
```

El array $_FILES['imagen']['error'] especifica por qué no se ha podido subir el archivo, lo que permite especificar un mensaje de vuelta para cada tipo de error. Devuelve un integer con el número de error:

Error	**Valor**	**Significado**
UPLOAD_ERR_OK	0	No hay errores
UPLOAD_ERR_INI_SIZE	1	Supera el tamaño máximo indicado en php.ini
UPLOAD_ERR_FORM_SIZE	2	Supera el tamaño máximo indicado en MAX_FILE_SIZE de html
UPLOAD_ERR_PARTIAL	3	Sólo se ha subido el archivo parcialmente
UPLOAD_ERR_NO_FILE	4	No se ha subido ningún archivo
UPLOAD_ERR_NO_TMP_DIR	6	Falta la carpeta temporal
UPLOAD_ERR_CANT_WRITE	7	No se puede escribir en el directorio especificado
UPLOAD_ERR_EXTENSION	8	Una extensión de PHP ha detenido la subida

11.4.3 Subida Múltiple Archivos

Una subida múltiple de archivos en PHP en la práctica se hace normalmente con ayuda de JavaScript. Una buena librería para subir archivos es DropzoneJS.

En el formulario HTML lo único que cambia es que se añade un atributo multiple y que en el atributo name del input file se añade corchetes para indicar que es un array:

```
<input type="file" name="imagenes[]" multiple="multiple" />
```

En cuanto al backend, omitiendo validaciones esta puede ser una forma de hacerlo:

```
$directorioSubida = "uploads/";
$max_file_size = "51200";
if(isset($_POST["submit"]) && isset($_FILES['imagenes'])){
    $nombres = $_FILES['imagenes']['name'];
    $temporales = $_FILES['imagenes']['tmp_name'];
    $tipos = $_FILES['imagenes']['type'];
    $errores = $_FILES['imagenes']['error'];
    // Iteramos sobre los arrays creados
    for ($i = 0; $i < count($temporales); $i++){
        if(move_uploaded_file($temporales[$i],
$directorioSubida.$nombres[$i])){
            echo "Se ha subido {$nombres[$i]} correctamente <br>";
        } else {
            echo "Ha habido algún error al subir algún archivo";
        }
    }
}
```

11.5 Seguridad de Sesiones

Aunque con las sesiones y las cookies no se pueda quebrantar la seguridad de la aplicación de forma directa, mediante el robo de sesiones se pueden comprometer las cuentas de los usuarios, y si éstos tienen permisos especiales las consecuencias pueden ser peor de lo esperado.

La mayoría de las veces PHP guardará una cookie en el ordenador del cliente llamada PHPSESSID (puede cambiarse al nombre que se desee) cuando se usen sesiones. Esta cookie guardará un valor, un identificador de sesión, que está asociado con algún tipo de datos en el servidor. Si el usuario tiene una session ID válida, los datos asociados con la sesión se incluirán en el superglobal array $_SESSION. Las sesiones pueden también transferirse a través de URL. En ese caso sería algo como _?PHPSESSID=id_aqui_.

La sesión ID funciona de forma parecida a la llave de un cajón en un banco, con la que puedes acceder a lo que sea que haya en ese cajón. La llave de tu cajón puede ser robada al igual que puede serlo la sesión ID de usuarios (robada o interceptada).

Cuando el atacante roba una session ID y trata de usarla para entrar en la aplicación como si fuera el verdadero usuario se le llama session hijacking. Cuando el atacante establece la session ID para la sesión de un usuario se denomina session fixation. Estos ataques no se pueden evitar en su totalidad pero se pueden tomar medidas para prevenirlos.

El problema de la seguridad en sesiones aumenta cuando se utiliza un hosting compartido, es decir, usar el mismo servidor que otros usuarios. En un server Linux, por defecto las sesiones de guardan en el directorio /tmp, que guarda archivos temporales y ha de ser legible y escribible para todo el mundo. Si tus sesiones se guardan ahí, otros usuarios podrían encontrar tus datos.

11.5.1 Opciones de configuración

La administración de sesiones HTTP es una parte fundamental de la seguridad web. Algunas de las configuraciones más importantes que pueden manipularse en PHP son las siguientes:

- _session.cookielifetime = 0. El cero tiene un significado especial, le dice a los navegadores que no guarden una cookie permanentemente. Así pues, si se cierra el navegador, la session ID se borra inmediatamente. Se desaconseja variar este valor, y si se quiere una aplicación con autologin, existen alternativas mucho más seguras.
- _session.usecookies = On y _session.use_onlycoockies = On. Aunque las HTTP cookies tienen algunos problemas, es la forma más recomendable de manejar sesiones.
- _session.use_strictmode = On. Esto previene al módulo de sesión iniciar una sessión ID sin inicializar. El módulo de sesión sólo acepta session ID válidas generadas por él mismo, rechazando cualquier sesson ID proporcionada por usuarios. Sessión ID injection puede hacerse con cookie injection a través de JavaScript. Se recomienda mantelerla en On.
- _session.cookiehttponly = On. Rechaza el acceso a la cookie de sesión vía JavaScript. Esto previene el robo de cookies a través de JavaScript injection. Se puede usar session ID como CSRF protection key, pero no es recomendable.
- _session.cookiesecure = On. Permite el acceso a la cookie session ID sólo cuando el protocolo es HTTPS. Si tu aplicación es sólo HTTPS, es recomendable tener activada esta opción.
- _session.gcmaxlifetime = [elegir el menor posible]. Número de segundos tras los cuales los datos serán considerados basura y limpiados con posterioridad. Garbage Collection puede ocurrir al inicio de sesión mediante probabilidad. Esta opción no garantiza la eliminación de sesiones antiguas. Aunque el desarrollador no ha de fiarse del todo de esta configuración, se recomienda ajustarla al menor valor posible. Es mejor ajustar _session.gc_probability y session.gc_divisor_ para que sesiones obsoletas se eliminen con la frecuencia apropiada.
- _session.use_transsid = Off. El tener esta opción de manejo de session ID transparentes desactivada mejora la seguridad de sesiones eliminando la posibilidad de session ID injection y session ID leak. De todas formas, pueden emplearse session ID transparentes si es necesario.
- _session.refererrcheck = [tu url original]. Si está activala la opción anterior, session.use_trans_sid, el uso de esta opción es recomendable, ya que reduce el riesgo de ID injection.
- _session.cachelimiter = nocache. Asegúrate de que los contenidos HTTP no son cacheados para sesiones autentificadas. "private" suele usarse para cuando no hay

ningún dato de seguridad en el contenido HTTP, y "public" cuando no contiene ningún dato privado en general.

- _session.hashfunction = "sha256". Una función hash más fuerte generará una session ID más fuerte. Cuanto más complejo sea el cifrado mejor: sha384, sha512...

El módulo de sesión no puede garantizar que la información guardada en una sesión sólo sea vista por el usuario que creó la sesión. Es necesario tomar más medidas para proteger totalmente la confidencialidad de la sesión.

Existen diferentes formas de que se filtre una session ID exitente a terceros. Si se produce una leaked session ID, posibilita a la tercera persona acceder a todos los recursos asociados con esa ID. Formas de que esto ocurra:

- URLs llevando una session ID. Si enlazas a un sitio externo desde una URL que contiene una session ID, ésta aparecerá en el referrer log del sitio externo.
- Un atacante más activo puede estar atento al tráfico de red. Si éste no está encriptado, las session IDs pueden viajar en texto plano por la red. La solución es implementar SSL en el servidor y hacerlo obligatorio a los usuarios. Se debería usar HSTS.

Cuando la opción _session.use_strict_mode_ está On, una session ID no iniciada se rechaza y se crea una nueva session ID. Esto protege ataques que fuerzan a los usuarios a usar una session ID. Por ejemplo, un atacante puede pasar URLs que contienen una session ID: http://example.com/page.php?PHPSESSID=23411. Si session.use_transsid está activado, la víctima iniciará la sesión con la session ID proporcionada por el atacante. session.use_strict_mode_ reduce el riesgo, aunque no asegura.

Para la autentificación de usuarios es muy recomendable añadir _session_regenerate_id()_, y debe ser llamada antes de establecer información de autentificación a $_SESSION. Esta función se asegura de que nuevas sesiones contienen información de autentificación almacenada sólo en una nueva sesión.

La session ID debería ser regenerada por lo menos cada vez que el usuario es identificado. No se ha de confiar en la expiración de la session ID. Los atacantes pueden acceder a las session ID de las víctimas periódicamente para evitar la expiración. Es recomendable implementar algún sistema propio para manejar sesiones antiguas.

_session_regenerate_id() por defecto no elimina una sesión antigua, y puede estar disponible para su uso. Para ello ha de ser destruída, añadiendo el parámetro delete_old_session_ TRUE a la función, aunque esto puede tener consecuencias inesperadas. Una sesión puede ser destruída cuando hay conexiones simultáneas a la aplicación o la red es inestable. En lugar de destruir la sesión antigua inmediatamente, se puede establecer un tiempo corto de expiración en $_SESSION. Si el usuario intenta acceder a la sesión antigua, deniega el acceso a ella.

No se deben usar session IDs muy duraderas para autologin porque incrementa el riesgo de robo de sesión. Una autologin key debe protegerse lo más posible, para ello se pueden utilizar atributos /httponly/.

11.5.2 Prevenir ataque session fixation

Cuando el atacante establece la session ID para la sesión de un usuario se denomina session fixation. Una vez que el atacante proporciona una URL al usuario con la session ID establecida y éste accede, el atacante, al conocer la session ID con la que se ha accedido, puede hacerse pasar por el usuario. Para prevenir el session fixation se deben tomar las siguientes medidas:

- Establecer _session.use_trans_sid = 0_ en el php.ini. esto le dice a PHP que no incluya el session ID en la URL, y no leer la URL en busca de identificadores.
- Establecer _session.use_only_cookies = 1_ en el php.ini. Esto le dice a PHP que nunca use URLs con session IDs.
- Regenerar la session ID siempre que el estado de la sesión cambie. Ejemplos: autentificación de usuario, guardar información importante en la sesión, cambiar cualquier cosa de la sesión...

11.5.3 Prevenir ataque session hijacking

Session hijacking es cuando el atacante roba una session ID y trata de usarla para entrar en la aplicación como si fuera el verdadero usuario. Como el atacante tiene el session ID, el servidor no puede distinguir cual es el verdadero usuario. Para prevenir el session hijacking se deben tomar las siguientes medidas:

- Usar un identificador hash muy potente. Directiva _session.hashfunction en php.ini. Lo ideal es _session.hashfunction = sha256 o _session.hashfunction = sha512.
- Enviar un hash potente. Directiva _session.hash_bits_percharacter en php.ini. Configura esta opción a _session.hash_bits_percharacter = 5. Es una traba más para cuando el atacante trate de adivinar el session ID. El ID será más corto pero usará más caracteres.
- Establece una entropía adicional con _session.entropyfile y _session.entropylength en php.ini. Por ejemplo _session.entropyfile = /dev/urandom y _session.entropylength = 256, el número de bytes que serán leídos del archivo entropy.
- Cambia el nombre por defecto de la sesión PHPSESSID. Este valor se establece con la función session_name(), con el valor de identificación propio como parámetro, antes de llamar a _session_start()_.
- Rotar el nombre de la sesión, pero ten en cuenta que todas las sesiones serán invalidadas si cambias esto, por ejemplo si se configura de forma que dependa en el tiempo.
- Rota el session ID a menudo. No es recomndable hacerlo en cada request (a no ser que se necesite un nivel de seguridad extremo), pero si en intervalos aleatorios. Si el atacante hace session hijacking no se desea que pueda utilizar una sesión durante demasiado tiempo.
- Incluir el user agent desde $_SERVER['HTTP_USER_AGENT'] en la sesión. Cuando la sesión comience, guardala en _$_SESSION['user-agent']_. Entonces en cada request posterior comprueba que coincide. Este valor puede ser falso, por lo que no es 100% seguro, pero es mejor que nada.
- Incluir la IP de usuario de $_SERVER['REMOTE_ADDR'] en la sesión. Cuando la sesión comience, guárdala en _$_SESSION['remote_ip']_. Esto puede ser problemático para

ciertos ISPs que usan direcciones IP múltiples para sus usuarios. Pero si se usa puede ser mucho más seguro. La única forma de que un atacante pueda falsear la IP es comprometiendo la red en algún punto entre el usuario real y tu servidor. Y si un atacante consigue comprometer una red, puede hacer cosas mucho peores que session hijacking.

- Incluir un token en la sesión y en el navegador que incrementas y comparas frecuentemente. Para cada request, haz $_SESSION['counter']++ en el lado del servidor. Crea algo en JS en el lado del navegador haciendo lo mismo (usando almacenamiento local). Entonces cuando se envía un request, coge el nonce de un token y verifica que el nonce es el mismo que en el servidor. Haciendo esto, se puede detectar una session hijacked ya que el atacante no tendrá el número exacto, y si lo tiene habrá dos sistemas transmitiendo el mismo número. Esto no funcionará en todas las aplicaciones, pero es otra manera de protegerse.

11.5.4 Diferencia entre session hijacking y session fixation

La diferencia entre session fixation y session hijacking reside simplemente en cómo la session ID es comprometida. En session fixation, el identificador se establece a un valor que el atacante conoce de antemano. En session hijacking se adivina o se roba del usuario. Una vez que el session ID es robado, los efectos de ambos dos son los mismos.

11.5.5 Ejemplo documentado

Esta función hará que tu script de inicio de sesión sea mucho más seguro. Hará que los crackers dejen de acceder al cookie de identificación de la sesión con JavaScript (por ejemplo en un ataque XSS). A su vez, la función “session_regenerate_id()”, la cual regenera la identificación de la sesión en cada carga de la página, ayudará a prevenir un robo de sesión. Nota: si vas a usar HTTPS en tu aplicación de inicio de sesión, configura la variable “$secure” a “verdadero”. En un ambiente de producción, será esencial que emplees HTTPS.

Crea un nuevo archivo llamado “functions.php” en el directorio “includes” de tu aplicación y agrégale el código a continuación.

```
<?php
include_once 'psl-config.php';

function sec_session_start() {
    $session_name = 'sec_session_id';   // Configura un nombre de sesión
personalizado.
    $secure = SECURE;
    // Esto detiene que JavaScript sea capaz de acceder a la identificación de
la sesión.
    $httponly = true;
    // Obliga a las sesiones a solo utilizar cookies.
    if (ini_set('session.use_only_cookies', 1) === FALSE) {
        header("Location: ../error.php?err=Could not initiate a safe session
(ini_set)");
        exit();
    }
    // Obtiene los params de los cookies actuales.
    $cookieParams = session_get_cookie_params();
    session_set_cookie_params($cookieParams["lifetime"],
```

```
        $cookieParams["path"],
        $cookieParams["domain"],
        $secure,
        $httponly);
    // Configura el nombre de sesión al configurado arriba.
    session_name($session_name);
    session_start();            // Inicia la sesión PHP.
    session_regenerate_id();    // Regenera la sesión, borra la previa.
}
```