Práctica 1. Registro de un usuario

Taller de desarrollo seguro con PHP

Mayo 2019

1. Objetivo de la práctica

El alumno deberá asegurar el sitio que permite la creación de un nuevo usuario a través de programación segura en PHP. Se deberá proteger el componente mediante las siguientes técnicas:

- Validación de datos.
- Sentencias preparadas.
- Cifrado de la contraseña al guardar en la base.
- Sanitizacion de datos que se usan en la página de bienvenida.
- Creación y verificación de tokens anti CSRF

2. Validación de datos

Para este caso en específico tenemos los siguietes datos a validar: nombre, apellido, email y contraseña. Lo primero que se debe hacer es saber si los datos que se han recibido están vacíos o no. Para poder saberlo se puede hacer uso de funciones que proporciona PHP como *isset()* y *trim()*. La primera de ellas nos ayuda a determinar si una variable está definida y no es NULL. La segunda elimina los espacios, muy útil cuando se envían campos llenos solamente de caracteres en blanco. A continuación un ejemplo:

Listing 1: Datos requeridos

```
function requerido($valor)
{
    $trimmedValue = trim($valor);
    return isset($trimmedValue) && $trimmedValue !== "";
}
```

Por otra parte, es de gran interés saber si cada uno de los datos trae el valor que esperamos. Por ejemplo, el campo de email debe contener un email válido. Para eso podemos hacer uso de la función *filter_var()*. Esta función la provee PHP por defecto y su función es filtrar una variable con el filtro que se indique. La lista de filtros se puede consultar en este sitio

Listing 2: Validación email

```
//Se revisa que sea un email valido
if (!filter_var($data['email'], FILTER_VALIDATE_EMAIL)) {
    return false;
}
```

Muchas veces se requiere el uso de una expresión regular para la validación de datos. Se puede crear una o hacer uso de alguna que se haya creado en la comunidad. Por ejemplo, en el sitio regexr se pueden encontrar múltiples expresiones regulares generadas por la comunidad.

3. Sentencias preparadas

Una sentencia preparada o parametrizada nos ayuda a evitar ataques de inyección de código SQL debido a que los parámetros vinculados no necesitan ser escapados porque nunca son sustituidos directamente dentro del string de consulta. De acuerdo con la documentación de PHP hay un flujo de trabajo básico: la preparación y la ejecución. En la etapa de preparación se envía una plantilla de sentencia al servidor de bases de datos. El servidor realiza una comprobación de sintaxis e inicializa los recursos internos del servidor para su uso posterior. Durante la ejecución el cliente vincula los valores de los parámetros y los envía al servidor. El servidor crea una sentencia desde la plantilla de la sentencia y los valores vinculados para ejecutarla usando los recursos internos previamente creados.

Listing 3: Comparación entre sentencias

```
/* Sentencia no preparada */
$sql = "INSERT␣INTO␣usuarios␣(nombre,␣apellido,␣email,␣password)
VALUES␣('$nombre',␣'$apellido',␣'$email',␣'$password')";
$resultado =  $mysqli->query($sql)

/* Sentencia preparada */
$sql='INSERT␣INTO␣usuarios␣(nombre,␣apellido,␣email,␣password)␣VALUES␣(?,␣?,␣?,␣?)';
$data = $mysqli->prepare( $sql );
$data->bind_param( 'ssss', $nombre, $apellido, $email, $password);
$resultado = data->execute();
```

En la vinculación de las variables se coloca como primer parametro el caracter que define qué tipo de valor tendrá dicha variable. Existen cuatro tipos a elegir:

- **i** la variable correspondiente es de tipo entero
- **d** la variable correspondiente es de tipo double
- **s** la variable correspondiente es de tipo string
- **b** la variable correspondiente es un blob y se envía en paquetes

Para más información consultar la sección de consultas preparadas en la documentación de PHP

4. Cifrado de contraseña

Los datos sensibles siempre deben almacenarse de manera cifrada en la base de datos. En caso de que sucediera un compromiso de la base, se dificulta la posibilidad de divugar o tener acceso a esta información.

Para lograr el cifrado de datos PHP nos provee de la función *password_hash()* la cual se emplea para crear un hash con una cadena dada utilizando el algoritmo más fuerte actualmente disponible.

Listing 4: Cifrado de contraseña

```
$sql = 'INSERT INTO usuarios (nombre, apellido, email, password) VALUES (?, ?, ?, ?)';
$data = $mysqli->prepare( $sql );
$password = password_hash($password,PASSWORD_DEFAULT);
$data->bind_param( 'ssss', $nombre, $apellido, $email, $password);
if ($data->execute()) {
    $mensaje = "Gracias por registrarte $nombre $apellido";
}else{
    $errorMessage = "Ha ocurrido un error al guardar los datos.";
}
```

Para más información del método se puede consultar su documentación

A continuación se muestra un ejemplo de cómo se guarda el Hash de la contraseña.

p@p.com	Hola123.
p@p.com	$2y$10$u6ar0PfCKp.5IjgZ1V60g.V2757fL5GfPRpratsPE6I...

5. Sanitización de datos

Otra forma de proteger una aplicación web es através de la *sanitización* de datos. Este proceso consiste en la eliminación o escapado de caracteres

no deseados en la información que se recibe. Se utiliza principalmente para evitar ataques XSS y de inyección de código SQL.

PHP nos provee de funciones que nos permiten sanear diferentes tipos de cadenas de información. A continuación se muestran las funciones que se utilizarán:

- *htmlentities()*. Convierte todos los caracteres aplicables a entidades HTML
- *mysqli_real_escape_string()*. Escapa los caracteres especiales de una cadena para usarla en una sentencia SQL. Evita errores en caso de estar presentes estos caracteres.

Para ver el catálogo completo de funciones que proporciona PHP, se puede aceder a su documentación a través de la siguiente liga

Listing 5: Sanitización de datos

```
//Saneado de datos
//Codificacion de caracteres html
$email = htmlentities($email);
$nombre = htmlentities($nombre);
$apellido = htmlentities($apellido);
```

6. Creación y verificación de tokens anti CSRF

Para la creación de un token anti CSRF PHP provee funciones que, en su conjunto, permiten la generación de un token aleatorio de tamaño definido. Además, es recomendable darle tiempo de vida a cada token que se genera. A continuación un ejemplo:

Listing 6: Creación token

```
function generateSessionToken() {
        if( isset( $_SESSION[ 'csrf_token' ] ) ) {
                unset( $_SESSION[ 'csrf_token' ] );
        }
        $_SESSION[ 'csrf_token' ] = md5(uniqid(rand(), TRUE));
        $_SESSION['csrf_token_time'] = time();
}
```

Por otra parte, es necesaria la validación del token una vez se recibe información. Se recomienda que la validación contemple tanto la validez del token, así como el tiempo en el que expira. Para hacerlo se realiza lo siguiente:

Listing 7: Validación token

```
function csrf_token_is_valid() {
```

```
        if(isset($_POST['csrf_token'])) {
                $user_token = $_POST['csrf_token'];
                $stored_token = $_SESSION['csrf_token'];
                return $user_token === $stored_token;
        } else {
                return false;
        }
}
```

Listing 8: Validación de expiración token

```
function csrf_token_is_recent() {
        $max_elapsed = 60 * 60 * 24; // 1 dia
        if(isset($_SESSION['csrf_token_time'])) {
                $stored_time = $_SESSION['csrf_token_time'];
                return ($stored_time + $max_elapsed) >= time();
        } else {
                // Eliminar token expirado
                destroy_csrf_token();
                return false;
        }
}
```

Referencias

[1] https://www.php.net/manual/es/