eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Sistemas de Gestión de Seguridad de Sistemas de Información

Pentesting

Grado en Ingeniería Informática de Gestión y Sistemas de Información

Tercer curso

Francisco Fernández Condado
Diego González Tamayo
Xabier Unzilla Higuero
Abdessamad El Harbouli

Curso 2024-25

Índice de contenidos

Introducción

Este proyecto propone la realización de un examen de penetración o *Pentesting* sobre el proyecto de un Sistema Web realizado en la entrega anterior, disponible en GitHub para su consulta. Haciendo uso de la herramienta *ZAP*[1], que actúa como si fuera un proxy y examina todas las peticiones que se hacen al sitio web, examinaremos las posibles vulnerabilidades existentes en la página y, posteriormente, trataremos de corregirlas con la finalidad de hacerlo lo más seguro posible.

Comenzaremos realizando una primera auditoría de seguridad y, a partir de la información proporcionada por la herramienta, iremos identificando las diferentes vulnerabilidades detectadas. Posteriormente, realizaremos nuevas auditorías con el fin de ver si efectivamente las vulnerabilidades han sido o no parcheadas de la manera adecuada.

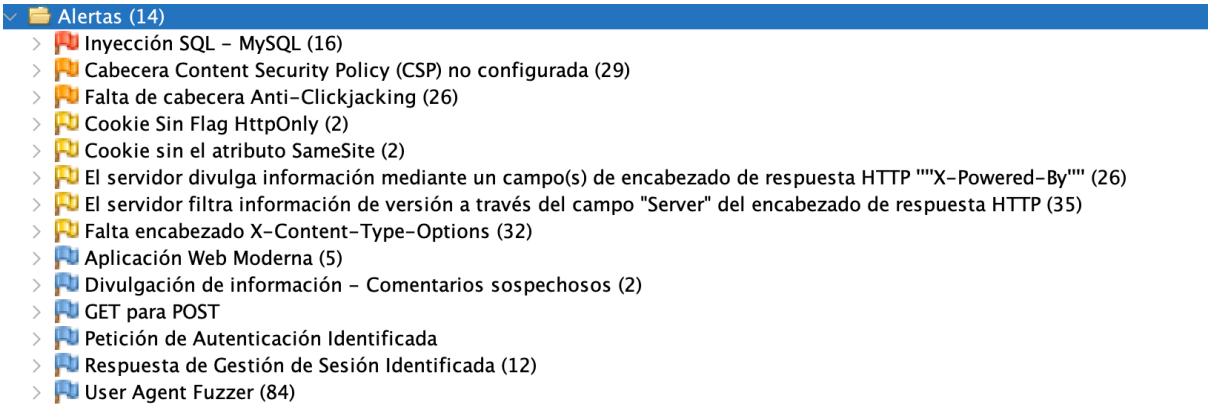
Auditoría inicial

En primer lugar, será necesario tener habilitado el contenedor de Docker asignado al proyecto, al igual que sucedía con la primera entrega. Los pasos a seguir son los mismos: asumiendo que se ha instalado previamente en el equipo la herramienta Docker[2] (incluyendo *docker-compose*), se debe acceder a la ruta en la que se haya clonado mediante *git* el repositorio y, a continuación, ejecutar el comando `docker-compose up -d`. Una vez hecho esto, será necesario acceder a la ruta `localhost:8890`, en la que está disponible la herramienta *phpMyAdmin* y, con ella, importar la base de datos[3] del fichero `database.sql`. Con esto, el proyecto estará disponible a través de la dirección URL `localhost:81`.

Para comenzar la auditoría, deberemos abrir la aplicación *ZAP* y, a continuación, escribir en la entrada de la URL a atacar la dirección `http://localhost:81`[4]. Indicaremos al sistema que utilice tanto el *spider tradicional* como el *spider ajax*, en este caso con *Firefox Headless*. A continuación, presionando el botón “Atacar” comenzará el análisis de vulnerabilidades.

[1] Puede obtenerse información adicional sobre la herramienta *ZAP* en su página web oficial.
[2] Si aún no se ha instalado, puede obtenerse a través de su sitio web.
[3] En la documentación de phpMyAdmin puede encontrarse información adicional sobre cómo utilizar esta funcionalidad que puede resultar ser de ayuda: https://docs.phpmyadmin.net/en/latest/import_export.html
[4] En caso de error, puede indicarse también la dirección `http://127.0.0.1:81`, que a efectos prácticos será la misma

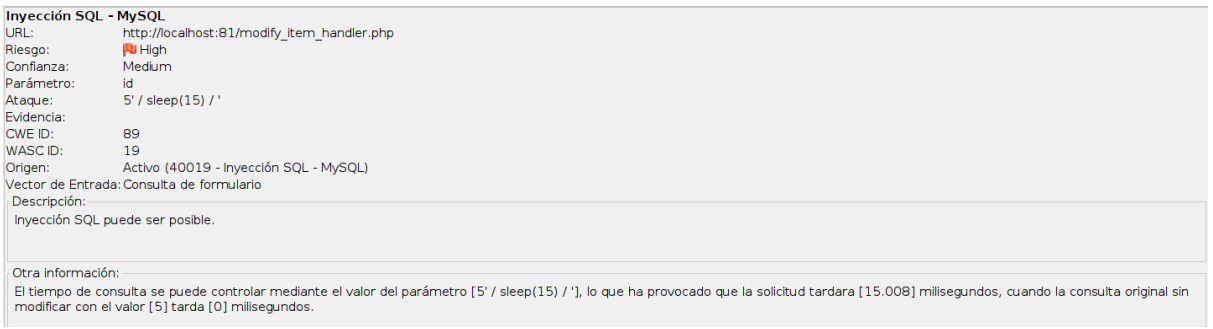
El análisis tardará unos minutos y, una vez completado, nos devolverá una lista de vulnerabilidades:



El detalle completo de la auditoría realizada se encuentra en el fichero *inicial.html* del directorio *auditorias* del repositorio, donde se especifica cada una de las amenazas, las direcciones URL afectadas y las referencias. A continuación, se irá analizando cada una de las alertas recibidas con el fin de buscar una solución.

Inyección SQL

El primer problema identificado es una vulnerabilidad de inyección SQL basada en el tiempo, en la que un atacante puede alterar un parámetro para modificar la consulta SQL y ejecutar código malicioso, así como el uso de la función `SLEEP()` en SQL. Esto permite a los atacantes identificar las vulnerabilidades de la aplicación mediante la observación de los tiempos de respuesta, que retrasa la respuesta del servidor.



Para evitar este tipo de ataques, es necesario validar y sanitizar adecuadamente los datos de entrada y utilizar consultas preparadas. Además, hay que añadir algunos detalles adicionales para fortalecer la seguridad.

El uso de consultas preparadas evita que los valores de entrada del usuario puedan alterar la estructura de la consulta SQL. En una consulta preparada, la estructura de la consulta SQL se define con antelación, y los datos del usuario se pasan por separado en una segunda etapa. Esto asegura que los datos se traten como valores y no como código SQL. El código modificado es el siguiente:

```
// Preparar consulta SQL para evitar inyección
$query = "INSERT INTO alimentos (nombre, fcompra, fcaducidad, calorias, precio) VALUES (?, ?, ?, ?, ?)";
$stmt = $conn->prepare($query);
```

La interrogación en la consulta actúa como un marcador de posición para cada valor de los datos del formulario, sin permitir que el usuario modifique la consulta.

El uso de la función bind_param se usa para asociar los datos del usuario a los marcadores de posición (?) de manera segura. Esta función también se asegura de que cada valor está correctamente escrito, reduciendo el riesgo de inyección. El *“sssdi”* define el tipo de dato de los parámetros de la consulta:

- “s” para string(cadena), en este caso es usado en nombre, fcompra y fcaducidad
- “d” para double(decimal), en este caso es usado en precio
- “i” para integer(entero), en este caso es usado en calorías

```
// Asignar los parámetros
$stmt->bind_param("sssdi", $nombre, $fcompra, $fcaducidad, $calorias, $precio);
```

Estos cambios protegen la integridad de la base de datos y aseguran que el sistema no sea vulnerable a modificaciones en las consultas por entradas manipuladas.

Estos cambios se aplican en diferentes ficheros del formulario, en este caso, en los ficheros que realizan respuestas a una base de datos (los handlers):

- add_item_handler.php
- delete_item_handler.php
- login_handler.php
- modify_item_handler.php
- modify_user_handler.php
- register_handler.php

Configuración de cabecera Content Security Policy (CSP)

La cabecera Content Security Policy o CSP es la encargada de indicar a los navegadores los contenidos que son seguros o no dentro de una página web. De este modo, pueden evitarse ataques de diversos tipos, como los de secuestro de contenidos o la inyección de contenido inseguro en las páginas, conocido como cross-site scripting (XSS). Adicionalmente, definir una política CSP en las cabeceras de las páginas puede ayudar a prevenir las técnicas de clickjaking y, aunque no lo aplicaremos en este proyecto, a definir que todas las páginas se carguen mediante el protocolo seguro HTTPS.

Para solucionar este problema, deberemos colocar en el fichero `.htaccess`.

```
<IfModule mod_headers.c>
     Header set Content-Security-Policy "default-src 'self';
script-src 'self'; style-src 'self'; img-src 'self'; font-src
'self'; connect-src 'self'; form-action 'self'; frame-ancestors
'none';"
</IfModule>
```

Esta cabecera debería ser lo suficientemente segura. No obstante, para que funcione, es necesario también modificar el *Dockerfile* con el fin de habilitar el módulo *headers* de Apache.

Sin embargo, al ejecutar el cambio, vimos que las alertas que se mostraban en la primera versión dejaban de funcionar, y rebajar la política CSP no era una opción. Como solución, decidimos incorporar nuevos ficheros PHP para gestionar los errores en lugar de mostrar las alertas mediante JavaScript en línea. De este modo, la integración es más segura.

Configuración de cabecera anti-clickjacking

En línea con la cabecera anterior, también resulta interesante configurar la cabecera X-Frame-Options. En la actualidad, el uso de esta cabecera consiste en indicar a los navegadores cuándo puede embedirse contenido del sitio, algo comúnmente utilizado para ataques de tipo clickjacking. Existen diversos modos de configurar esta cabecera en función de cómo funcione la aplicación web. Sin embargo, como en este caso no queremos permitir

ningún tipo de *embed* por la simple naturaleza de la misma, la configuraremos con el tipo “deny”.

Para ello, añadiremos una nueva línea en el fichero de configuración *.htaccess*:

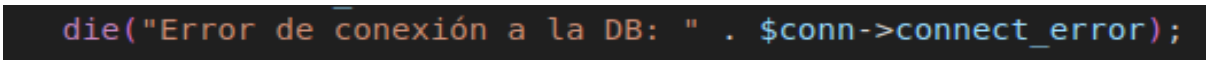
```
<IfModule mod_headers.c>
    Header always set X-Frame-Options "DENY"
</IfModule>
```

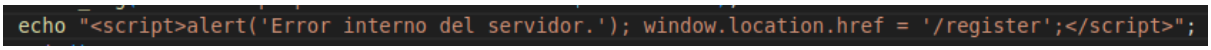
No obstante, ZAP seguía reportando el error tras añadirla en algunas páginas, que casualmente correspondían con una respuesta 404 de la página. Como solución, configuramos una nueva plantilla, *404.php*, que muestra una página básica de error 404 y permite volver a inicio. De este modo, al enviarse contenido como en cualquier otra página, Apache añade automáticamente la cabecera.

Divulgación de error de aplicación

Este error de ZAP indica que el código podría revelar información sensible (como la ubicación del archivo o el mensaje de error del servidor) en caso de una excepción o fallo de consulta. Para ello, se debe mejorar la gestión de errores para evitar que el usuario vea los detalles internos del sistema, los cuales podrían ser usados en ataques futuros.

En el código actual, se usa la función `die()` para mostrar errores de conexión con detalles específicos al usuario en la base de datos. Esto podría revelar información sobre el sistema y la configuración de la base de datos. A su vez, el código utiliza mensajes como “Error interno del servidor”, que podrían ser problemáticos si son usados en exceso, ya que muestran mensajes de errores específicos al usuario.





Para solucionar el problema, por un lado se usan mensajes de errores genéricos de cara al usuario, esto es, en lugar de mostrar el mensaje de error real, muestra un mensaje genérico que no revele detalles sobre el sistema. Por otro lado, se utiliza la función `error_log()` en cada punto de error, para registrar detalles en un log interno, sin mostrarlos al usuario.

```php
error_log("Error de conexión a la base de datos: " . $conn->connect_error);
echo "<script>alert('Error de conexión. Por favor, inténtalo más tarde.'); window.location.href = '/register';</script>";
exit();
```

```php
error_log("Error al preparar la consulta de inserción: " . $conn->error);
echo "<script>alert('Error interno del servidor. Por favor, inténtalo más tarde.'); window.location.href = '/register';</script>";
exit();
```

Supresión del encabezado de respuesta HTTP ""X-Powered-By""

El encabezado X-Powered-By puede revelar información sobre la tecnología o el software que está utilizando el servidor, en este caso PHP. Esta información puede ayudar a los atacantes a identificar posibles vulnerabilidades relacionadas con la tecnología revelada, así como detalles sobre el lenguaje o framework en el que está construida la aplicación o la versión específica de PHP.

Para solucionarlo, estableceremos un nuevo valor en el fichero de configuración *.htaccess* a nivel general del servidor, que se encargará de desactivar la cabecera X-Powered-By, ocultando por tanto la información pertinente:

```
<IfModule mod_headers.c>
   Header unset X-Powered-By
</IfModule>
```

Configuración del encabezado X-Content-Type-Options

El encabezado X-Content-Type-Options es usado por el servidor para indicar los tipos de MIME que se especifican en los encabezados Content-Type, siendo por tanto necesario configurarlo para evitar posibles ataques de *sniffing*. Para ello, de nuevo modificaremos el fichero *.htaccess* de configuración del servidor, especificando en el mismo la política a seguir y solicitando añadir una nueva cabecera en cada página servida:

```
<IfModule mod_headers.c>
   Header set X-Content-Type-Options "nosniff"
</IfModule>
```

Evitar que el servidor muestre información a través del campo “Server” de la cabecera HTTP

Si comprobamos la cabecera “Server” de una solicitud HTTP, podremos ver que el servidor se está encargando de especificar, además del uso de Apache, la versión y el sistema operativo de la máquina del contenedor Docker: *Apache/2.4.25 (Debian)*. Esto es algo que puede resultar peligroso, por ejemplo si aparece una vulnerabilidad en una versión específica, ya que podría hacer que los atacantes le saquen partido.

Para evitar que esto suceda, a nivel del fichero *.htaccess* deshabilitamos la cabecera:

```
<IfModule mod_headers.c>
   Header unset Server
</IfModule>
```

Sin embargo, por la configuración de Apache, continúa enviando información. Para solucionarlo, debemos añadir en la configuración general tanto el campo *ServerTokens Prod*, *ServerSignature Off* y, de forma opcional, en nuestro caso hemos optado por configurar también *LogLevel crit* para evitar que se muestren datos adicionales en los mensajes de error. Esto lo haremos modificando el fichero *Dockerfile* que se encarga de instanciar el proyecto, añadiendo las siguientes líneas.

```
RUN echo 'ServerTokens Prod' >> /etc/apache2/apache2.conf && \
   echo 'ServerSignature Off' >> /etc/apache2/apache2.conf && \
   echo 'LogLevel crit' >> /etc/apache2/apache2.conf
```

Con ello, solo se revela que se hace uso de Apache en la cabecera, pero no la versión ni el sistema operativo del servidor.

Configuración correcta de cookies

Nuestro portal web utiliza cookies para gestionar las sesiones de PHP y el inicio de sesión de usuarios. No obstante, hasta el momento se hacía uso de la configuración por defecto de dicho motor, lo que hacía que no se definiesen parámetros como *SameSite*, que define que la cookie únicamente se utilizará para este portal, o el flag *HttpOnly*. En primer lugar, solventamos este error sobre nuestras cookies propias usadas para gestionar el inicio de sesión de los usuarios, tal y como se puede apreciar en la siguiente imagen:

```php
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['enviar'])) {
    $email = htmlentities($_POST['email']);

    // Configuración de la cookie con 1 hora de duración
    setcookie('email', $email, time() + 3600, "/; SameSite=Strict", "", true, true);

    // Redirigir a index.php
    header("Location: index.php");
    exit();
}
```

Y, en lo que respecta a las cookies de sesión genéricas de PHP, en este caso se debían configurar los mismos parámetros editando el fichero de configuración *php.ini*. Sin embargo, en nuestro proyecto, no disponemos de dicho archivo. Para facilitar la configuración, hemos añadido la siguiente línea al *Dockerfile*:

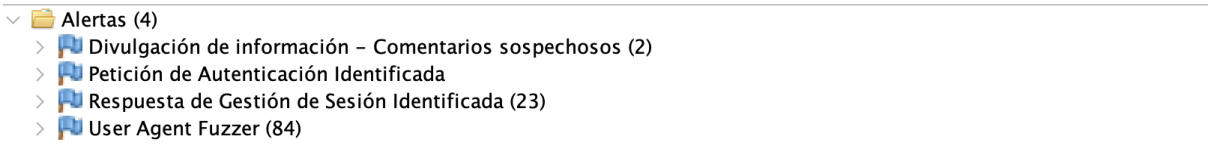
```
RUN echo 'session.cookie_samesite=Strict' >>
/usr/local/etc/php/conf.d/session.ini
```

Y, por otro lado, también se ha añadido la siguiente línea que define la misma configuración de cara al servidor basado en Apache:

```
echo 'Header always edit Set-Cookie (.*) "$1; SameSite=Strict"' >>
/etc/apache2/apache2.conf
```

Auditoría final

Tras las modificaciones mencionadas y diversas pruebas entre cada una de ellas, hemos parecido encontrar la configuración óptima de acuerdo a las especificaciones de ZAP, tal y como se muestra a continuación:



En este caso, las 4 alertas que se muestran son únicamente informativas, en gran parte procedentes de la realización del análisis por parte de la propia herramienta. El detalle completo de la auditoría realizada se encuentra en el fichero *final.html* del directorio *auditorias* del repositorio, donde se especifica cada una de las amenazas, las direcciones URL afectadas y las referencias. Tal vez lo más relevante guarde relación con los comentarios sospechosos que se indican en la primera alerta. No obstante, tras una

comprobación, vemos que se relaciona con parte del código de las librerías utilizadas por Bootstrap, no representando por tanto una amenaza real para la aplicación web.

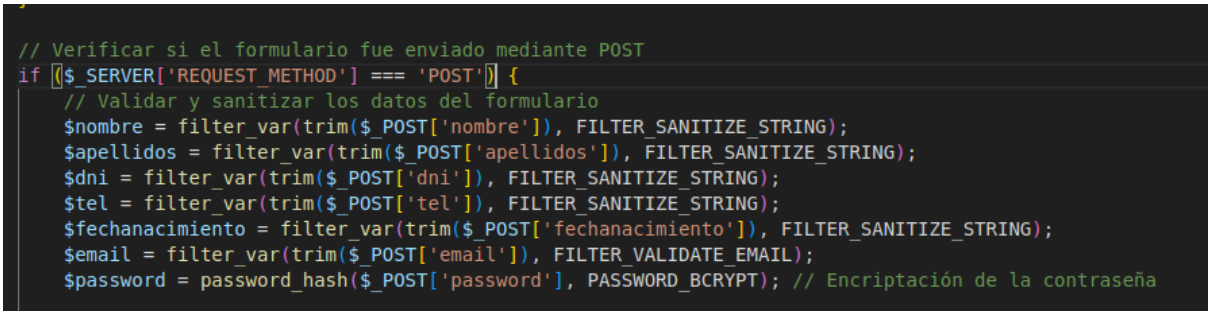
Correcciones adicionales realizadas

A pesar de no haber sido reconocidas mediante amenazas por parte de la herramienta *ZAP*, hemos optado por realizar las siguientes correcciones que incrementan la seguridad del proyecto:

Verificación de contraseñas mediante hash

Antes, las contraseñas se guardaban en la base de datos como texto plano, lo cual era inseguro ya que los datos de los usuarios eran fácilmente accesibles. Ahora, se almacena un hash para cada una de ellas en la base de datos, lo que hace que los datos sean más inaccesibles. Para arreglar esto se han modificado los siguientes ficheros de formularios:

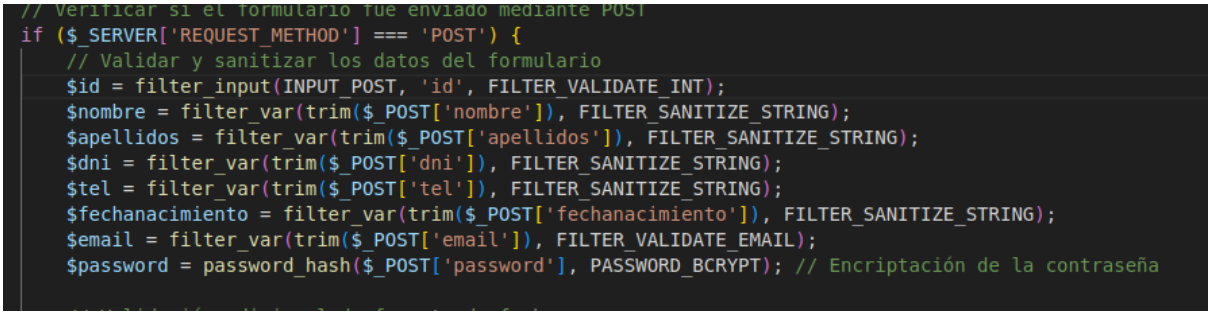
- Register_Handler:
    - Código nuevo:



```php
// Verificar si el formulario fue enviado mediante POST
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Validar y sanitizar los datos del formulario
    $nombre = filter_var(trim($_POST['nombre']), FILTER_SANITIZE_STRING);
    $apellidos = filter_var(trim($_POST['apellidos']), FILTER_SANITIZE_STRING);
    $dni = filter_var(trim($_POST['dni']), FILTER_SANITIZE_STRING);
    $tel = filter_var(trim($_POST['tel']), FILTER_SANITIZE_STRING);
    $fechanacimiento = filter_var(trim($_POST['fechanacimiento']), FILTER_SANITIZE_STRING);
    $email = filter_var(trim($_POST['email']), FILTER_VALIDATE_EMAIL);
    $password = password_hash($_POST['password'], PASSWORD_BCRYPT); // Encriptación de la contraseña
```

Mediante la última línea de la imagen, se genera el *hash* para la contraseña introducida, que será guardado en una variable `password` para su posterior introducción en la base de datos junto a los demás datos del usuario a registrarse.

- Modify_User_Handler:
    - Código nuevo:



```php
// Verificar si el formulario fue enviado mediante POST
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Validar y sanitizar los datos del formulario
    $id = filter_input(INPUT_POST, 'id', FILTER_VALIDATE_INT);
    $nombre = filter_var(trim($_POST['nombre']), FILTER_SANITIZE_STRING);
    $apellidos = filter_var(trim($_POST['apellidos']), FILTER_SANITIZE_STRING);
    $dni = filter_var(trim($_POST['dni']), FILTER_SANITIZE_STRING);
    $tel = filter_var(trim($_POST['tel']), FILTER_SANITIZE_STRING);
    $fechanacimiento = filter_var(trim($_POST['fechanacimiento']), FILTER_SANITIZE_STRING);
    $email = filter_var(trim($_POST['email']), FILTER_VALIDATE_EMAIL);
    $password = password_hash($_POST['password'], PASSWORD_BCRYPT); // Encriptación de la contraseña
```

Funciona prácticamente igual que el fichero Register_Handler, en el cual se registra en una variable password el hash correspondiente a la contraseña para luego modificar la base de datos.

- Login_Handler
    - Código nuevo

```php
// Verificar si el formulario fue enviado mediante POST
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Validar y sanitizar el email y la contraseña
    $email = filter_var(trim($_POST['email']), FILTER_VALIDATE_EMAIL);
    $password = $_POST['password'];

    // Verificación básica del email y la contraseña
    if (!$email || empty($password)) {
        echo "<script>alert('Por favor, introduce un correo y una contraseña válidos.'); window.location.href = '/login';</script>";
        exit();
    }
```

En este formulario la contraseña se trata de una manera diferente. Primero, la contraseña del formulario se pone en una variable con nombre contraseña. Después, se comprueba si se ha introducido la contraseña.

```php
    // Verificar si se encontró un usuario con el email proporcionado
    if ($result->num_rows > 0) {
        // Obtener los datos del usuario
        $user = $result->fetch_assoc();

        // Verificar la contraseña encriptada
        if (password_verify($password, $user['password'])) {
            // Guardar los datos del usuario en la sesión
            $_SESSION['id'] = $user['id'];
            $_SESSION['user_email'] = $user['email'];

            echo "<script>alert('¡Bienvenido! Has iniciado sesión correctamente'); window.location.href = '/';</script>";
        } else {
            echo "<script>alert('La contraseña introducida es incorrecta.'); window.location.href = '/login';</script>";
        }
    } else {
        echo "<script>alert('El usuario no existe. Por favor, revisa tu correo y vuelve a intentarlo.'); window.location.href = '/login';</script>";
    }
```

Por último, mediante un *if*, se comprueba que el hash de la contraseña almacenada en la base de datos, que se había extraído previamente mediante un *query*, coincide con el hash de la contraseña que ha introducido el usuario.

Log de inicios de sesión

Es importante saber quién se ha conectado, ha intentado conectarse o cuantas veces se ha intentado conectar en un tiempo determinado al servidor. Mediante un registro de inicios de sesión, se almacenan estos intentos con el fin de contar con una política de monitorización de quién ha intentado entrar al servidor web y así poder bloquear o avisar al usuario al que pertenezca la cuenta.

```php
// Registrar el inicio de sesión en LogsUsuarios
$logQuery = "INSERT INTO LogsUsuarios (id, correo, FechaHoraConexion, conectado) VALUES (?, ?, NOW(), 1)";
$logStmt = $conn->prepare($logQuery);

if ($logStmt) {
    $logStmt->bind_param('ss', $user['id'], $user['email']);
    $logStmt->execute();
    $logStmt->close();
} else {
    error_log("Error al registrar el log: " . $conn->error);
```

Se ha creado una nueva tabla en la base de datos, *LogsUsuarios*, que se utilizará para almacenar el ID del usuario, el correo electrónico, un timestamp y si el inicio de sesión es o no correcto. Después podrán consultarse las tablas para obtener toda la información, que se suma a los *logs* ya existentes tanto de Apache como de MariaDB sobre las consultas realizadas.

Especificación de versiones de los componentes

Aunque para el resto de componentes se especifica la versión específica, el gestor phpMyAdmin se instala con la versión *latest* según el fichero *docker-compose.yml*. Por tanto, se descarga la última versión disponible en Docker Hub y no conocemos cuál es. Hemos optado por modificar la imagen por *phpmyadmin:5.2.1*. De este modo, en caso de haber algún tipo de problema, conoceremos la versión exacta utilizada.

Conclusiones y trabajo futuro

Mediante la realización de este *pentesting*, hemos podido hacer frente a grandes vulnerabilidades de seguridad como el SQL Injection, el uso de cabeceras y configuraciones de Apache correctas, así como la verificación de los parámetros de las cookies con el fin de prevenir nuestro sistema ante posibles ataques, pudiendo poner en práctica nuestros conocimientos.

No obstante, a continuación se indican algunos puntos que aún quedan pendientes de mejora para conseguir que el sistema sea lo más seguro posible:

- **Pruebas en servidor real con tráfico HTTPS**: para este proyecto, no se ha optado por configurar un certificado SSL dado que se está trabajando en local y no se cuenta con un dominio personalizado en el que pueda conseguirse un certificado que firme una AC y que haga seguras las comunicaciones. No obstante, si se usa en un entorno real, conviene configurar este aspecto.

- **Actualización de componentes**: actualmente, se hace uso de versiones antiguas de MariaDB y de PHP. Aunque por motivos de compatibilidad se han mantenido o se han variado únicamente de forma ligera, convendría actualizar todos los componentes para evitar posibles vulnerabilidades.
- **Gestión de contraseñas**: en el fichero *docker-compose.yml*, se establecen algunas contraseñas de la base de datos en texto plano. Convendría hacer uso de los secretos para poder evitar que la información quede expuesta. Asimismo, de cara a los usuarios, habría que configurar una política de contraseñas más eficiente, solicitando cambiarla cada cierto tiempo y verificando que se cumplen una serie de parámetros para ver si es segura. Adicionalmente, una solución tipo reCAPTCHA podría ser una buena idea para evitar posibles ataques de fuerza bruta.
- **Política de copias de seguridad**: teniendo en cuenta que la base de datos se irá actualizando de forma periódica, convendría establecer una buena política de copias de seguridad con el fin de prevenir posibles desastres y actuar lo más rápido posible.
- **Gestión de logs y política de monitorización**: aunque se han incluido herramientas para gestionar los logs, convendría establecer una buena política de monitorización para evitar posibles ataques, preferentemente con intervención automática.

Bibliografía

Este proyecto se basa en el repositorio *docker-lamp de mikel-egaña-aranguren*, que propone un stack LAMP en Docker. Es importante destacar que se ha realizado uso de Bootstrap v5.3.3 para agilizar las tareas de diseño, basándose en su licencia MIT. A continuación se citan los documentos, sitios web y herramientas consultadas durante la realización del proyecto:

- *Docker Docs*
- *Cloning a repository / GitHub*
- *Import and Export / phpMyAdmin*
- *LAMP / Wikipedia*
- *Bootstrap Docs*
- *HTML / mdn web docs*
- *Cómo validar DNI español con JavaScript / local Horse*
- *regex101*
- *PHP Form Handling / W3Schools*
- *ChatGPT / OpenAI*
- *Copilot / Microsoft*
- *Bootstrap Examples*
- *Bootstrap buttons / bootstrap temple*
- *Getting Started with Writing and Formatting / GitHub*
- *Extended Syntax / Markdown Guide*
- *ZAP Proxy*
- *Content Security Policy (CSP) / mdn web docs*
- *X-Frame-Options / mdm web docs*
- *X-Content-Type-Options / mdm web docs*
- *How to Hide Your Apache Version and Linux OS From HTTP Headers / Inmotion Hosting*
- *PHP setcookie "SameSite=Strict"? / Stack Overflow*