

Sistemas de Gestión de Seguridad de Sistemas de Información

Grado en Ingeniería Informática de Gestión y
Sistemas de Información

Sistema Web: Auditoría de Seguridad
2024-2025

Fecha:

Bilbao, a X de Noviembre de 2024

Autores:

Larrazabal Subinas, Asier

Blasco Balaguer, Aritz

García Puebla, Ainhoa

García Coloma, Diego

Martín Paniagua, Marcos

Cotano Romero, Aitor

Índice

1. Introducción.....	3
2. Auditoría mediante OWASP ZAP (Zed Attack Proxy).....	4
3. Análisis de resultados OWASP ZAP	9
3.1. Redirección de HTTP A HTTPS.....	10
3.1.1. Cookie No HttpOnly Flag Low 1 && Cookie without SameSite Attribute Low 1:..	10
3.1.2. Pasos de redirección de tráfico http a https:.....	11
4. Rotura de control de acceso.....	13
4.1. Descripción.....	13
4.2. Posibles causas.....	13
4.2.1. Falta de autenticación Requerida.....	13
4.2.2. Roles y Permisos Mal Implementados.....	15
4.2.3. Dependencia en el lado del cliente.....	15
4.2.4. Nombres de usuario repetidos.....	17
4.2.5 Accesos fallidos.....	18
4.2.6 Exceso de peticiones a la API.....	19
5. Fallos Criptográficos.....	22
5.1. Descripción.....	22
5.2. Posibles vulnerabilidades criptográficas y soluciones en nuestra aplicación web....	22
5.2.1. Almacenamiento inseguro de contraseñas.....	22
5.2.2. Fallas en la generación de números aleatorios.....	22
5.2.3 Implementación insegura de algoritmos criptográficos.....	23
5.2.4 Canales inseguros para la transmisión de datos sensibles.....	24
5.2.5 Exposición de datos sensibles en registros o logs:.....	24
6. Inyección.....	28
6.1. Descripción.....	28
6.2. SQL Injection.....	28
6.2.1. Análisis mediante SQLMap.....	29
6.2.2. Prevención de la vulnerabilidad ¿Porqué nuestra página ya es segura?.....	30
6.3. XSS (Cross-Site Scripting).....	30
6.3.1. Análisis de la vulnerabilidad en “items.php”.....	31
6.3.2. Simulación de ataque XSS.....	31
6.3.3. Solución.....	33
7. Configuración y diseño inseguro.....	34
7.1. Descripción.....	34
7.2. Posibles causas.....	34
7.2.1. Imprimir mensajes de error que contengan información confidencial.....	34
7.2.2. Almacenamiento de credenciales sin protección.....	35
7.2.3. Violación de los límites de confianza.....	36
8. Componentes vulnerables y obsoletos.....	39

8.1. Descripción.....	39
8.2. Cómo prevenir.....	39
8.2. Posibles causas.....	39
8.2.1. Vulnerabilidades de PHPMailer.....	39
9. Fallos de identificación y autenticación.....	40
9.1. Descripción.....	40
9.2. Posibles causas.....	40
9.2.1. Ausencia o mal diseño de autenticación:.....	40
9.2.2. Contraseñas débiles o políticas de contraseñas inadecuadas:.....	41
9.2.3. Falta de autenticación multifactor (MFA):.....	42
9.2.4. Mal manejo de tokens de sesión:.....	45
9.2.5. Falta de protección contra ataques CSRF:.....	45
10. Fallos en la integridad de datos y software.....	48
10.1. Descripción.....	48
10.2. Cómo prevenirlos.....	48
10.3. Posibles causas.....	48
10.3.1. Manipulación de archivos críticos del sistema.....	48
10.3.2. Falta de verificación de integridad en las comunicaciones.....	49
10.4. Conclusión.....	50
11. Fallos en la monitorización de la seguridad.....	51
11.1. Descripción.....	51
11.2. Cómo prevenirlos.....	51
11.3. Posibles causas.....	51
11.3.1. Ausencia de supervisión de cambios sensibles.....	51
11.4. Conclusión.....	54
12. Referencias.....	55

1. Introducción

Este es el documento de la auditoría de seguridad realizada para el proyecto de la asignatura de Sistemas de Gestión de Seguridad de Sistemas de Información. En él se recogen todas las vulnerabilidades que se nos han pedido analizar:

- Rotura de control de acceso.
- Fallos criptográficos.
- Inyección.
- Diseño inseguro.
- Configuración de seguridad insuficiente.
- Componentes vulnerables y obsoletos.
- Fallos de identificación y autenticación.
- Fallos en la integridad de datos y software.
- Fallos en la monitorización de la seguridad

En este documento se registran las vulnerabilidades vistas en clase y otras que no se han visto en clase. Para realizar este trabajo hemos utilizado herramientas de pentesting ZAP y hemos estudiado las vulnerabilidades más comunes según el informe OWASP 2021.

Este trabajo se ha realizado usando como base la primera entrega de este proyecto. El repositorio para la segunda entrega se encuentra en el enlace: https://github.com/2001uri15/SGSSI-Proyecto/tree/entrega_2

2. Auditoría mediante OWASP ZAP (Zed Attack Proxy)

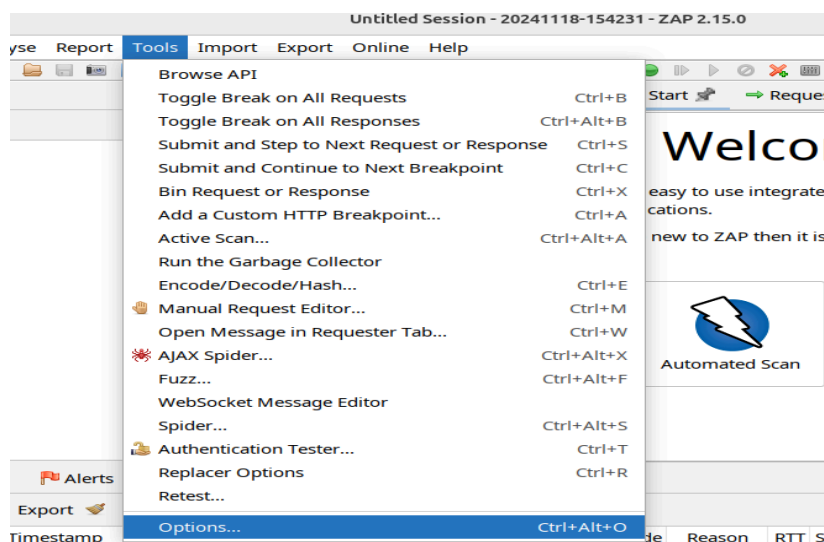
Una auditoría con dicha aplicación, nos ayuda a detectar vulnerabilidades para su posterior evaluación y reimplementación de código que lo mitigue o solucione.

Pasos a seguir:

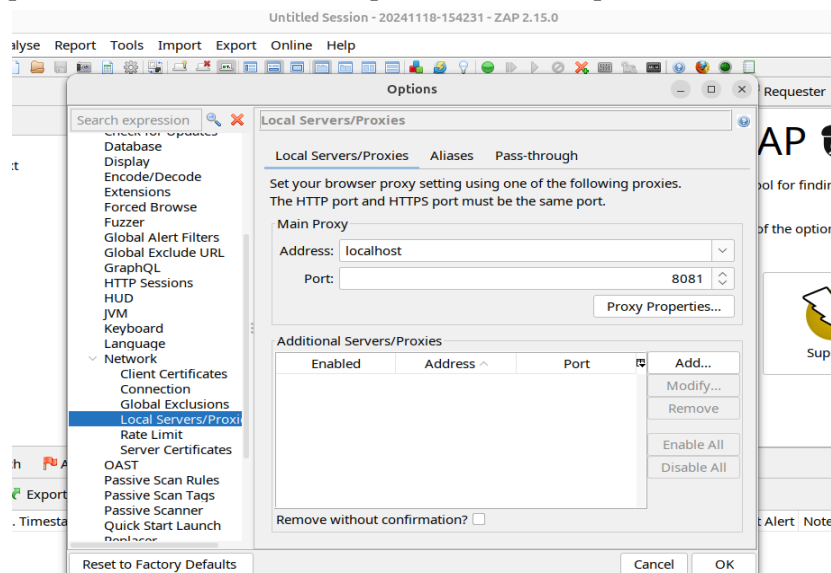
1- Suponemos que ya disponemos del programa ZAP en el sistema, pero en caso contrario, descargamos dicho software desde su página oficial [ZAP – Download](#), en donde seleccionamos “Linux Package”.

2- Aún con dicho proxy en nuestro sistema operativo, debemos configurar tanto el programa como nuestra web, para que OWASP ZAP sea capaz de interceptar el flujo de información del navegador para su posterior análisis.

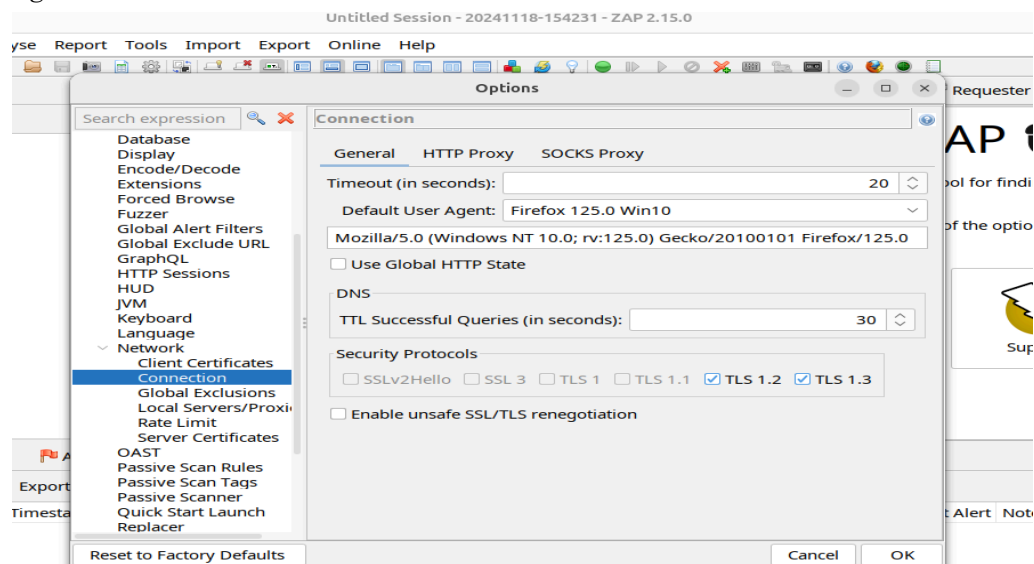
- En primer lugar, accedemos a OWASP ZAP, y en tools>options:



- Después, vamos a la pestaña Network>Local Servers/Proxies y configuramos de esta forma el puerto y el address, debido a que el puerto por defecto es el 8080, pondremos el 8081 para evitar problemas de puertos en uso:

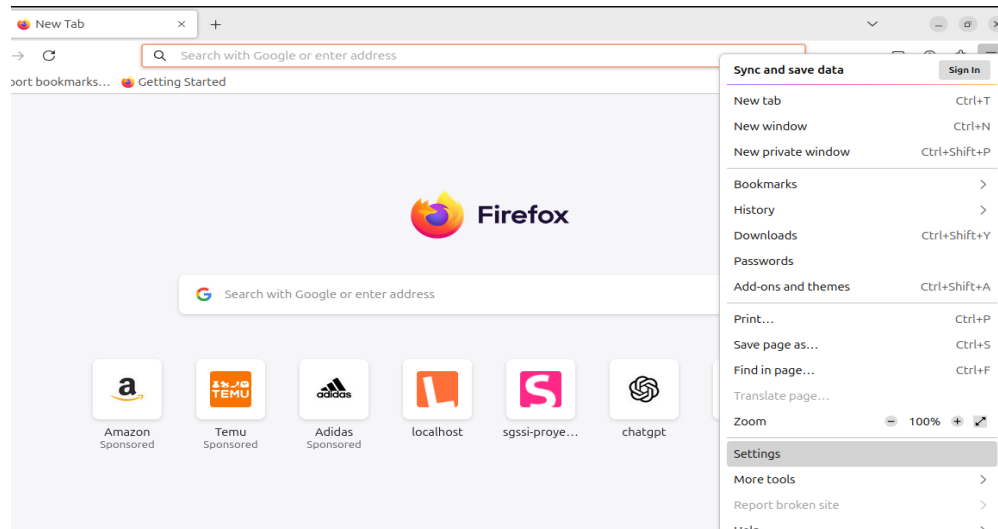


- Una vez configurado el puerto y el address, hay que asegurarse que nuestro navegador con el que haremos los accesos a nuestra página web esté configurado como el predeterminado y que OWASP ZAP esté configurado para usar el User Agent adecuado:

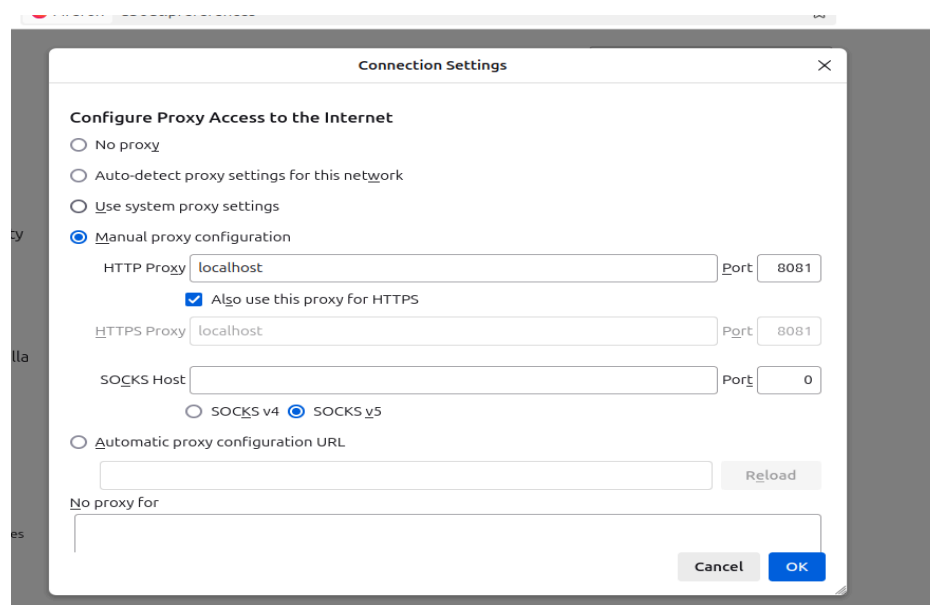


- Una vez estos pasos hayan sido realizados, pasamos a la configuración de nuestro navegador, Firefox en este caso:

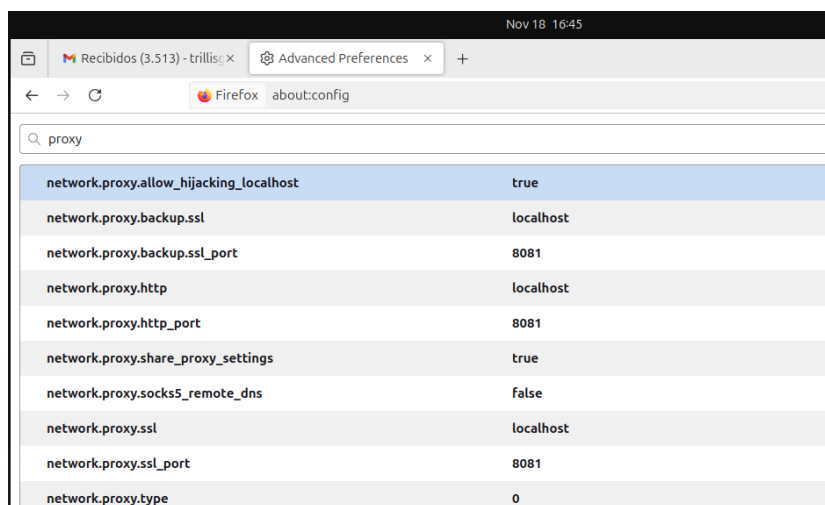
- Para empezar, nos vamos a la configuración de Firefox:



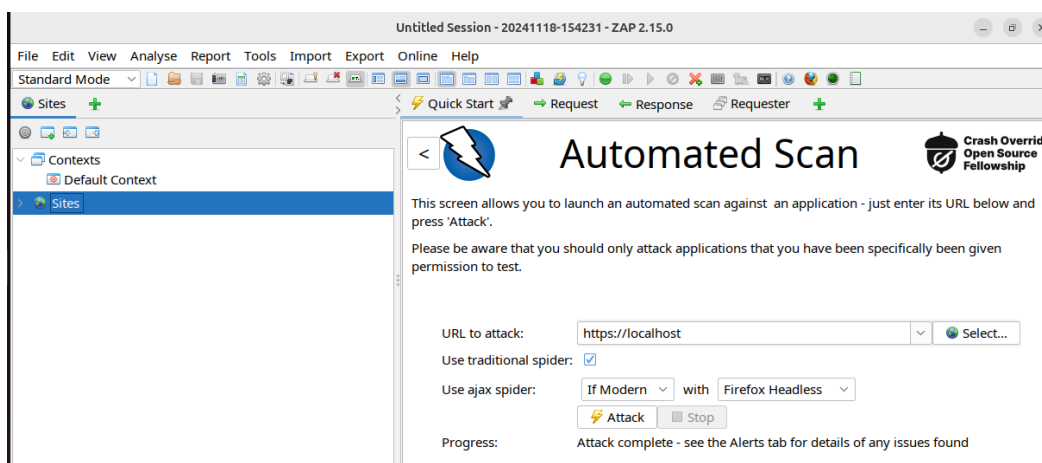
- En esta pestaña, seleccionamos “Network settings” que estará debajo del todo en la sección de “settings”, ahí necesitaremos configurar la proxy de esta forma:



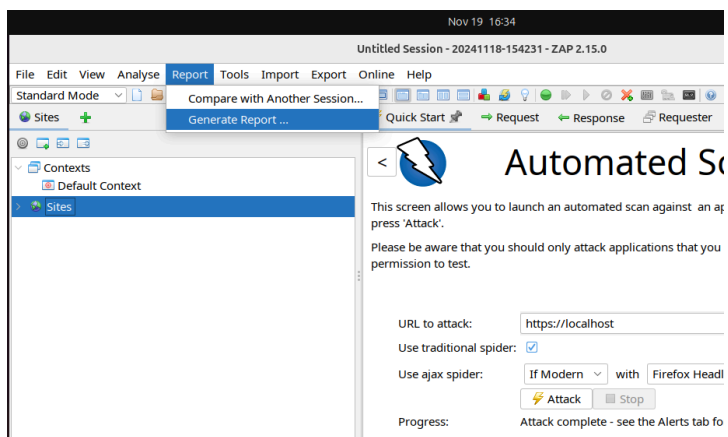
- A veces, según la configuración de la página web, es posible que por defecto estén bloqueadas las direcciones localhost y 127.0.0.1, lo cual hará que de ninguna forma nuestro ZAP reciba el flujo de información en dicha dirección. Para solventar esto, debemos hacer lo siguiente:
 - Escribimos en la barra superior de firefox “about:config” y aceptamos el riesgo
 - Luego, en el buscador de la propia sección, buscamos “network.proxy.allow_hijacking_localhost” y la ponemos a “true”.



- Una vez realizado estos cambios, procedemos a abrir nuestro OWASP ZAP e iniciar la auditoría, de forma que, en su pestaña principal seleccionamos “Automated scan”, y seleccionamos la dirección “<https://localhost>” que es donde está nuestra web:



- Por último seleccionamos Report>Generate Report:



De esta forma, crearemos un report de la página que queremos y se guardará en formato html en el repositorio de nuestra elección.

3. Análisis de resultados OWASP ZAP

La auditoría muestra flags a analizar, clasificadas según peligrosidad contra posibles ataques externos:

- Cross Site Scripting (Reflected)High 1
- PII Disclosure High 1
- Source Code Disclosure - CVE-2012-1823 High 2
- Absence of Anti-CSRF Tokens Medium 145
- Application Error Disclosure Medium 2
- CSP: Wildcard Directive Medium 54
- CSP: script-src unsafe-eval Medium 54
- CSP: script-src unsafe-inline Medium 54
- CSP: style-src unsafe-inlineMedium 54
- Content Security Policy (CSP) Header Not Set Medium 110
- Hidden File FoundMedium1
- Missing Anti-clickjacking Header Medium 40
- Vulnerable JS Library Medium 2
- Cookie No HttpOnly Flag Low 1
- Cookie without SameSite Attribute Low 1
- Information Disclosure - Debug Error Messages Low 3
- Private IP DisclosureLow3
- Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) Low 72
- Server Leaks Version Information via "Server" HTTP Response Header Field Low 236
- X-Content-Type-Options Header Missing Low 110
- Authentication Request Identified Informational 28
- CSP: X-Content-Security-Policy Informational 54
- CSP: X-WebKit-CSP Informational 54
- GET for POST Informational 1
- Information Disclosure - Sensitive Information in URL Informational 49
- Information Disclosure - Suspicious Comments Informational 172
- Loosely Scoped Cookie Informational 105
- Modern Web Application Informational 80
- Obsolete Content Security Policy (CSP) Header Found Informational 54
- Session Management Response Identified Informational 131
- User Agent Fuzzer Informational 456
- User Controllable HTML Element Attribute (Potential XSS) Informational 252

Para la mejora de nuestra web, debemos tener en cuenta los flags, comprenderlos y aplicar el cambio necesario para evitar posibles fallas de integridad en la misma:

3.1. Redirección de HTTP A HTTPS

3.1.1. Cookie No HttpOnly Flag Low 1 && Cookie without SameSite Attribute Low 1:

Vamos a establecer las cookies de seguridad precisas y necesarias antes de meternos en el código de nuestra web, para ello nos centraremos en estas dos flags, las cuales nos dicen que la falta de dichas sentencias supone un problema de tipo “low” en el sistema, aunque ciertamente creemos en la importancia de establecer dichos parámetros como es debido.

En primer lugar, `httpOnly` es un atributo de una cookie que la hace accesible solo mediante `http/https`, esto significa que no podrá ser leída ni modificada por scripts del lado del cliente (como `javaScript`) gracias a lo cual se impedirá que la misma sea manipulada o expuesta por scripts maliciosos. En este primer caso, `httponly=true` protege a la cookie contra ataques tipo Cross-Site Scripting (XSS).

En segundo lugar, también haremos uso del atributo `Secure`, el cual estipula que la cookie sólo podrá enviarse al servidor si la conexión es segura (`https`), esto significa que la cookie no será transmitida en solicitudes hechas mediante `http` no cifrado, lo que protege la información de posibles interceptaciones. Este atributo en particular mitiga el riesgo de ataques de secuestro de sesión (session Hijacking) o de técnicas como sniffing en redes públicas (wifi abierto).

En tercer lugar, también usaremos el atributo `Samesite`. Este atributo es una medida de seguridad aplicada a las cookies para controlar cuándo deben ser enviadas junto con solicitudes `HTTP`, dependiendo del origen de la solicitud. Ayuda a proteger las aplicaciones web contra ataques como Cross-Site Request Forgery (CSRF) y mejora la privacidad del usuario. En nuestro caso la configuraremos como “*Strict*”, lo cual define que la cookie sólo será enviada en solicitudes que provienen del mismo dominio que configuró la cookie, esto significa que si el usuario hace clic en un enlace de un sitio externo que redirige a nuestra web, la cookie no será enviada, esto es, solo se envían cookies en solicitudes generadas directamente desde el mismo dominio (navegando dentro del mismo).

Estos tres atributos tienen algo en común: en el formato en la que queremos colocarlas, Activas y `Samesite` a `Strict`, debemos hacer uso exclusivamente de `https`, ya que de otra forma dichos atributos no funcionarían correctamente. Es por ello, que necesitamos redirigir todo el tráfico de `localhost` en `http` (tanto el puerto 81 de nuestra aplicación web como el 8080 de `phpmyadmin`) a `https` y para lograr dicho fin, hemos usado `nginx`, que redirige las solicitudes al backend (web o `phpmyadmin`) y sirve como proxy inverso.

3.1.2. Pasos de redirección de tráfico http a https:

En primer lugar, hemos modificado el archivo docker-compose.yml para cargar un nuevo contenedor para nginx, y también hemos quitado los puertos de los contenedores phpmyadmin y web, para evitar que se puedan acceder mediante http.

docker.compose.yml:

```
nginx:
  image: nginx:latest
  container_name: nginx
  ports:
    - "443:443" # HTTPS
    - "81:80"   # HTTP, redirigido a HTTPS
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf
    - ./nginx/cert.crt:/etc/nginx/certs/cert.crt
    - ./nginx/cert.key:/etc/nginx/certs/cert.key

  depends_on:
    - web
    - phpmyadmin
  networks:
    - my-network
```

- ports:
 - 443:443: Mapea el puerto 443 (HTTPS) del host al puerto 443 del contenedor, permitiendo acceso HTTPS.
 - 81:80: Mapea el puerto 80 (HTTP) del contenedor al puerto 81 del host, redirigiendo el tráfico HTTP.
- volumes:
 - En estas líneas, montamos en el contenedor las claves SSL que vamos a crear para usar en nuestra aplicación de forma que funcione correctamente con el comando “openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout cert.key -out cert.crt” que colocaremos en /nginx para que sean leídas y montadas correctamente. Además, montaremos el archivo nginx.conf que será el que configure nginx, de forma que el tráfico de los dos puertos mencionados previamente sean redirigidos de forma correcta.

- depends_on:
 - Define las dependencias del servicio nginx. Esto asegura que los servicios web y phpmyadmin se inicien antes que nginx.

Con estos cambios, nos será posible acceder a nuestra web mediante <https://localhost>, pero aún no hemos añadido los parámetros necesarios para aumentar la seguridad de la misma. Para ello, añadiremos esta línea dentro del archivo nginx.conf:

```
proxy_cookie_path / "/; secure; HttpOnly; SameSite=Strict";
```

Esta línea modifica las cookies generadas por el backend y enviadas al cliente a través de Nginx. Añade los atributos Secure, HttpOnly, y SameSite=Strict a las cookies con la ruta /.

Una vez realizado este proceso, procederemos a aplicar los cambios en el código según los próximos pasos a proceder, teniendo en cuenta las diferentes vulnerabilidades dadas en clase, finalizando con un informe ZAP a posteriori para comprobar los resultados de nuestros cambios.

4. Rotura de control de acceso

4.1. Descripción

La rotura de control de acceso es una vulnerabilidad que ocurre cuando los mecanismos diseñados para restringir el acceso a datos o funciones sensibles no funcionan correctamente. Esto permite que un usuario (malintencionado o no) acceda a recursos o realice acciones para las cuales no está autorizado.

En sistemas seguros, el control de acceso asegura que los usuarios solo puedan acceder a los datos y funcionalidades que les corresponden según su rol o permisos. Cuando estos controles fallan o no están implementados adecuadamente, la seguridad del sistema queda comprometida.

4.2. Posibles causas

4.2.1. Falta de autenticación Requerida

Algunas rutas o acciones no verifican si el usuario está autenticado

Ejemplo:

- Un endpoint público permite acceder a datos sensibles sin requerir autenticación
- No se verifica `$_SESSION['loggedin']` en las páginas protegidas.

Solución:
















En caso de nuestra web, tenemos implementado una funcionalidad que, a pesar de que se muestren los coches de la lista sin haber iniciado sesión, a la hora de querer eliminar coches, editar o añadir se hace una comprobación de inicio de sesión, de forma que en el formulario de `items.php` tenemos ocultas estas dos opciones:

Coches						LogIn
Listado de Coches						
ID	Matrícula	Modelo	Marca	Tipo de Combustión	Color	#
2	1121-jns	M5	BMW	Gasolina	negro	
4	1121jng	M5	BMW	Gasolina	Blanco	
6	1821-jns	M5	BMW	Gasolina	Rojo	
7	7302-yaq	Seat	BMW	Gasolina	Rojo	
9	2121-aaa	Seat	BMW	Gasolina	Rojo	

visualización después de iniciar sesión:

Coches
Modificar Datos
Hola, diegogar420
Cerrar Sesión

Listado de Coches

ID	Matrícula	Modelo	Marca	Tipo de Combustión	Color	#
2	1121-jns	M5	BMW	Gasolina	negro	  
4	1121jnq	M5	BMW	Gasolina	Blanco	  
6	1821-jns	M5	BMW	Gasolina	Rojo	  
7	7302-yaq	Seat	BMW	Gasolina	Rojo	  
9	2121-aaa	Seat	BMW	Gasolina	Rojo	  

[Añadir Item](#)

Dicho resultado conseguimos gracias a la autenticación `$_SESSION['loggedin']` dentro del código ya existente en `items.php`:

```

<?php if (isset($_SESSION['username'])): ?>
    <a href="modify_item.php?id=<?php echo $coche['id']; ?>"><i class="bi bi-pencil-square"></i></a>

    <a href="delete_item.php?id=<?php echo $coche['id']; ?>"><i class="bi bi-trash-fill"></i></a>
<?php endif; ?>

, -----

<?php if (isset($_SESSION['username'])): ?>
    <div style="align: right;">
        <a href="add_item.php" >Añadir Item</a>
    </div>

```

4.2.2. Roles y Permisos Mal Implementados

Ejemplo:

- Un usuario con rol de “cliente” puede acceder a recursos o acciones destinadas sólo a “administradores”

Solución:

En nuestra web no tenemos divididos los usuarios según rol, ya que cada uno podrá acceder a los mismos puntos de la web que cualquiera (mientras se esté logueado).

De todas formas, si en un futuro quisieramos implementar una comprobación de rol antes de permitir la visualización de opciones avanzadas, podríamos gestionar un if tal que:

```
if ($_SESSION['role'] !== 'admin') {  
  
    header("Location: unauthorized.php");  
  
    exit();  
  
}
```

Para controlar así el acceso a la sección de código en caso de que el valor del rol de la sesión sea “admin”, además de implementar otro archivo .php como ejemplo, que redirija al usuario a una pestaña de error (unauthorized.php) en caso de no tratarse de un administrador.

4.2.3. Dependencia en el lado del cliente

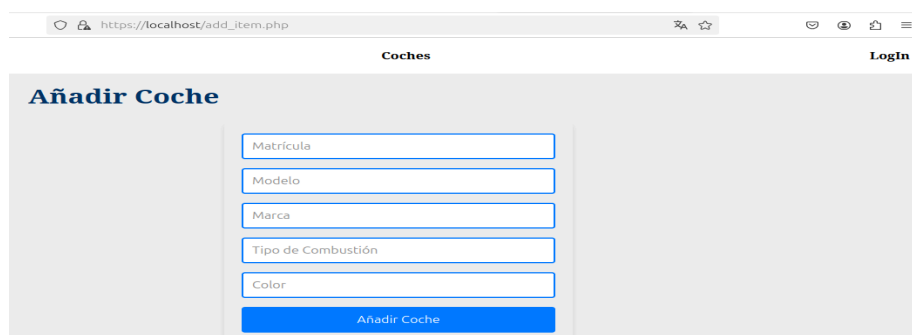
El sistema depende de controles en el frontend para restringir el acceso, como ocultar botones o menús.

Ejemplo:

- Aunque un botón "Eliminar" no se muestra al usuario, el endpoint /delete sigue accesible si el usuario lo invoca manualmente.

Solución:

Con la configuración actual de la web, la verificación de los botones ocultos sólo ocurre en el frontend, lo que no es suficiente para evitar que un usuario no identificado acceda a dichos recursos escribiendo en el backend /delete_items.php, como se muestra en la foto:



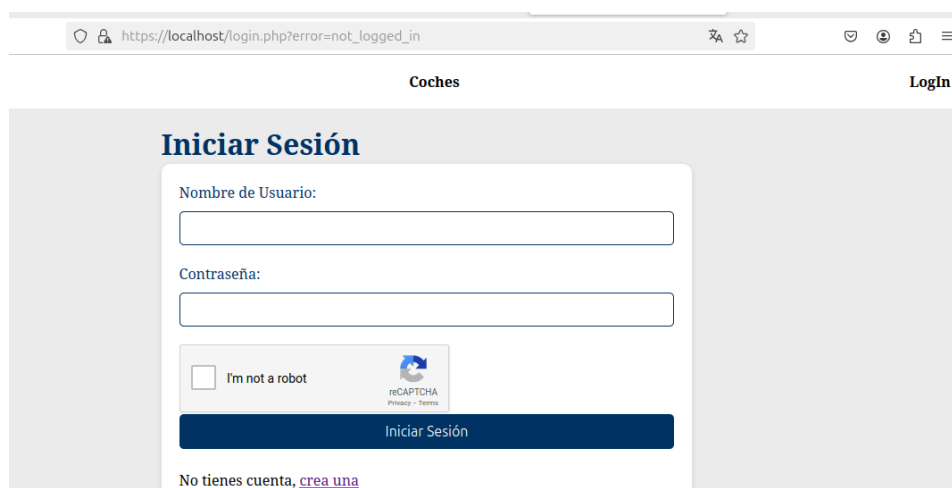
Para evitar esta rotura de acceso en el backend, debemos modificar los archivos que requieran de permisos especiales (estar logueado) para ser accedidos: “delete_item.php”, “modify_item.php” y “add_item.php”.

La modificación pertinente en estos casos, trataría de validar la sesión en cada uno de los archivos, añadiendo una configuración en los endpoints del backend:

```

<?php
session_start();
require_once 'plantillas/header.php'; // Incluimos el header
// Verificar si el usuario ha iniciado sesión
if (!isset($_SESSION['username'])) {
    header("Location: login.php?error=not_logged_in");
    exit();
}
  
```

En dichos 3 archivos especiales, donde solo teníamos la línea “*require_once ‘plantillas/header.php’;*”, ahora hemos añadido una verificación de sesión con una redirección a error de no logueo en caso de que no se den los necesarios casos para su debido acceso. Una vez implementado, en el caso de delete_item.php, si queremos acceder simplemente colocando tras localhost/ , delete_items.php, nos mostrará la redirección del error y directamente la pestaña de inicio de sesión.



4.2.4. Nombres de usuario repetidos

En este caso, se puede dar el caso de que el nombre de usuario elegido concuerde con otro ya en uso dentro de nuestra base de datos, lo cual podría dar fallos de rotura de acceso.

Solución:

Para dar con una solución, vamos a implementar una comprobación en el backend, dentro de *process_register.php*:

```

if ($stmt->num_rows > 0) {
    // Si el nombre de usuario ya existe, redirigir con un mensaje de error
    $_SESSION['error_message'] = "El nombre de usuario '$usuario' ya está en uso. Por favor, elige otro.";
    header("Location: register.php");
    exit();
}

```

De esta forma nos aseguramos de que el parámetro de entrada en usuario sea único dentro de nuestra base de datos de usuario. Además, vamos a implementar un cambio en el html de la página register.php, para que de esta forma se muestre un mensaje de error en el <body> del propio formulario, de esta forma:

```

<body>
  <div class="body-margin">
    <h1>Registro de Usuario</h1>
    <!-- Mostrar mensajes de error -->
    <?php
    if (isset($_SESSION['error_message'])) {
        echo "<p style='color: red;'>" . $_SESSION['error_message'] . "</p>";
        unset($_SESSION['error_message']); // Limpiar mensaje después de mostrarlo
    }
    ?>

```

Con estos cambios, ahora si queremos intentar registrarnos con un usuario ya en uso, por ejemplo “jon” en este caso, nos saldrá lo siguiente cuando le demos a “submit”:



Habiendo implementado esta útil solución, también sería conveniente añadir un segmento de código parecido para atrapar el caso en el que se intente registrar con un gmail ya en uso, esto es debido que a la hora de iniciar sesión, no haya problemas a la hora de enviar al gmail del usuario el código de verificación. Este código extra se vería así en process_register.php:

```
// Verificar si el correo electrónico ya existe
$stmt = $conn->prepare("SELECT id FROM usuarios WHERE mail = ?");
$stmt->bind_param("s", $email);
$stmt->execute();
$stmt->store_result();

if ($stmt->num_rows > 0) {
    // Si el correo electrónico ya existe, redirigir con un mensaje de error
    $_SESSION['error_message'] = "El correo electrónico '$email' ya está en uso. Por favor, usa otro.";
    header("Location: register.php");
    exit();
}
```

Con esta verificación, haciendo la prueba de registrar un usuario con el mismo gmail que uno ya dado de alta mostraría este error(para la prueba, se sabe que tenemos un usuario llamado diegogar420 con gmail trilliscg@gmail.com):



4.2.5 Accesos fallidos

Por ejemplo, cuando un usuario intenta iniciar sesión muchas veces y falla.

Solución:

En este caso la solución la desglosamos en las secciones de “Fallos de identificación y autenticación.” y “Fallos criptográficos.”, ya que en el primer caso, explicamos los cambios realizados para limitar los intentos de loguearse, de forma que después de 5, se bloquee la cuenta por x tiempo, y en el segundo apartado explicamos que logear los intentos de iniciar sesión y los fallos que han surgido es una buena práctica para alertar a los administradores de la web sobre posibles fallas tanto de inicio de sesión(importantes para nuestro apartado actual) como de posible filtración de datos sobre errores criptográficos presentes que sem muestre al exterior (en este caso, dentro de la sección “Fallos Criptográficos” explicamos las medidas tomadas para evitarlo).

4.2.6 Exceso de peticiones a la API

Una vulnerabilidad relacionada con la ausencia de limitaciones de la tasa de peticiones a una API (rate limiting) ocurre cuando no se controla la cantidad de solicitudes que un usuario o cliente puede realizar en un período de tiempo. Esta falta de control puede ser explotada por atacantes mediante programas automatizados(bots o scripts) para llevar a cabo diversas actividades maliciosas.

En nuestro caso, esto podría derivar en diversos ataques:

- Abuso de funcionalidades: funciones como el envío de correos electrónicos, generación de reportes o validación de códigos podrían ser abusados para causar gastos o saturar recursos.
- Denegación de Servicio(DoS) o Ataques distribuidos(DDoS): Un atacante(o múltiples bots en caso de DDoS) puede inundar la API con solicitudes hasta que consuma todos los recursos del servidor, haciéndola inaccesible para otros usuarios legítimos.
- Scraping y robo de contenido: Un bot puede extraer grandes cantidades de información de la web (como tablas enteras de la base de datos) en poco tiempo si no hay un límite.

Solución:

Para solventar este problema, tenemos implementado un Captcha de google, una herramienta poderosa que nos sirve para diversas funcionalidades:

- Ataques por fuerza bruta en el login: para proteger los accesos desde /login, debido a que a un bot le es difícil interactuar con esta herramienta.
- Creación masiva de cuentas falsas: los bots pueden explotar formularios de registro para crear cuentas falsas en masa, en este caso un captcha en el formulario de registro asegura que solo usuarios humanos puedan completar el proceso.
- Ataques de spam: los formularios de comentarios o contactos son objetivos comunes para spam automatizado. Un captcha reduce significativamente estas actividades.

Esta herramienta, aunque poderosa, no es suficiente para solucionar este problema, debido a que no protege de ataques tipo DoS y DDoS ni tampoco limita solicitudes legítimas excesivas(por ejemplo, un cliente mal diseñado) y afectar el rendimiento del sistema.

Por ello aplicaremos un “rate limiting” dentro de la configuración de nginx, esto es, dentro de la carpeta que montamos mediante docker-compose.yml (nginx.conf) donde tenemos los bloques de servidor(montado en /etc/nginx/conf.d/default.conf como previamente hemos comentado en los pasos de realización del proxy inverso) para gestionar la redirección de la web y de phpmyadmin de forma separada, definiendo diversos atributos que ya hemos explicado previamente.

En la sección “Location” de cada bloque de server, añadimos la siguiente línea:

```
location / {
    # Aplica rate limiting a toda la web
    limit_req zone=mylimit burst=20 nodelay;
```

Esto limita las peticiones a 20 por segundo, haciendo que las que excedan se devuelvan con un código de error.

Además de ello, crearemos un archivo de configuraciones que montaremos en /etc/nginx/nginx.conf (configurado este montaje en la sección nginx de nuestro docker-compose.yml, debajo de “volumes”), este se encargará de configurar correctamente el uso de dicho limit_req y su comportamiento (esta config no podemos montarla junto a los bloques de servidor ya que nos lanzaría este error en caso de que queramos ponerlo en el mismo archivo:

```
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/11/23 23:32:24 [emerg] 1#1: "user" directive is not allowed here in /etc/nginx/conf.d/default.conf:2
nginx: [emerg] "user" directive is not allowed here in /etc/nginx/conf.d/default.conf:2
```

Este archivo adicional de configuración de nginx quedaría de esta forma:

```
error_log /var/log/nginx/error.log;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    sendfile on;
    keepalive_timeout 65;

    limit_req_zone $binary_remote_addr zone=mylimit:10m rate=10r/s;

    include /etc/nginx/conf.d/*.conf;
}
```

Define comportamientos básicos antes del bloque http.

Explicación del funcionamiento del límite establecido:

- `limit_req_zone` : define una zona de memoria compartida (mylimit) donde se van a almacenar las IP que están siendo rastreadas para limitar las peticiones.
- `$binary_remote_addr` : este parámetro utiliza la IP del cliente como clave para rastrear las solicitudes, de esta forma tendremos control sobre cada IP registrada y será tratada de forma independiente.
- `10m`: Estableces un tamaño de memoria compartida de 10 MB, lo que es suficiente para rastrear aproximadamente 16.000 direcciones IP activas.
- `rate=10r/s`: Permite un máximo de 10 solicitudes por segundo para cada IP. si una IP supera ese límite, se bloquean las solicitudes adicionales.

Para comprobar esta funcionalidad, vamos a simular una cadena de 30 peticiones en 1 segundo, para ver si desde la número 21 nos las bloquea:

Ejecutamos: `for i in {1..30}; do curl -k -I https://web.localhost/; done` y observamos las salidas:

```

diego@Ubuntu:~/Downloads/SGSSI-Proyecto-main$ for i in {1..30}; do curl -k -I https://web.localhost/; done
HTTP/1.1 200 OK
Server: nginx/1.27.2
Date: Sat, 23 Nov 2024 23:53:15 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/7.2.2
Set-Cookie: PHPSESSID=895be2236fd17d36d6f31006d60054d7; path=/; secure; HttpOnly; SameSite=Strict
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
  
```

A la solicitud número 21, la salida cambia y nos reproduce 10 veces la siguiente salida:

```

HTTP/1.1 200 OK
Server: nginx/1.27.2
Date: Sat, 23 Nov 2024 23:53:15 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
X-Powered-By: PHP/7.2.2
Set-Cookie: PHPSESSID=0cdad838c95534612a98f746c4ecd649; path=/; secure; HttpOnly; SameSite=Strict
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache

HTTP/1.1 503 Service Temporarily Unavailable
Server: nginx/1.27.2
Date: Sat, 23 Nov 2024 23:53:15 GMT
Content-Type: text/html
Content-Length: 197
Connection: keep-alive

HTTP/1.1 503 Service Temporarily Unavailable
Server: nginx/1.27.2
Date: Sat, 23 Nov 2024 23:53:15 GMT
  
```

De esta manera, podemos observar que dicha implementación si es eficaz y funcional.

5. Fallos Criptográficos

5.1. Descripción

Los fallos criptográficos ocurren cuando se implementan mal los algoritmos criptográficos, se utilizan métodos inseguros o se manejan de manera incorrecta los datos sensibles, comprometiendo la seguridad del sistema.

En esta categoría, abordaremos la vulnerabilidad expuesta, para ello, primero debemos identificar las prácticas criptográficas que se están utilizando en nuestra aplicación web (por ejemplo, el manejo de contraseñas o datos sensibles) y asegurarnos de que se implementen correctamente.

5.2. Posibles vulnerabilidades criptográficas y soluciones en nuestra aplicación web

5.2.1. Almacenamiento inseguro de contraseñas

Si las contraseñas se almacenan en un texto plano o con un algoritmo débil (como MD5 o SHA1), un atacante que acceda a la base de datos podría obtener fácilmente las contraseñas.

Solución:

En nuestro sistema, tenemos esta sección vulnerable solucionada, ya que implementamos un hasheado de contraseña resistente a ataques de fuerza bruta, haciendo que su almacenado sea protegido por un algoritmo potente.

```
// Hashear la contraseña
$hashed_password = password_hash($password, PASSWORD_DEFAULT);
```

5.2.2. Fallas en la generación de números aleatorios

Uso de generadores predecibles en lugar de un CSPRNG (Cryptographically Secure Pseudo-Random Number Generator).

Solución:

A la hora de verificar el usuario en el login, generamos de forma aleatoria un número, lo cual puede ser fácil de adivinar, asique sería útil si usaremos algún comando de generación de código más compleja y segura:

En `process_login.php`, teníamos puesto a la hora de crear un código aleatorio:

```
$verification_code = rand(100000, 999999);
```

Siendo este código poco seguro e inapropiado para el caso. Por ello, hemos implementado una generación más compleja incluyendo además letras en él:

```
function generateVerificationCode($length = 6) {
    $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $code = '';
    for ($i = 0; $i < $length; $i++) {
        $code .= $characters[random_int(0, strlen($characters) - 1)];
    }
    return $code;
}

$email = $user['mail'];
$verification_code = generateVerificationCode(6);
$_SESSION['verification_code'] = $verification_code;
$_SESSION['user_email'] = $email;
```

Como podemos observar, usaremos de ahora en adelante el comando `random_int`, que es más seguro en este entorno de trabajo.

5.2.3 Implementación insegura de algoritmos criptográficos

Esto podría ocurrir si se usan funciones caseras o bibliotecas no probadas, pero como en nuestro caso no usamos codificación más allá de la contraseña, para la cual usamos `password_hash` con `PASSWORD_DEFAULT` que usa el algoritmo más seguro disponible en la versión de PHP (generalmente `bcrypt` o `Argon2`).

Por otro lado, `password_hash` genera automáticamente una sal aleatoria, lo que protege contra ataques de tablas arcoiris, por eso después de varias pruebas codificando la contraseña previamente y luego hasheandola, hemos determinado que no era necesario, el esfuerzo de generar claves aleatorias de sesión y usar funciones de descryptación y encriptación en muchos de los ficheros de gestión de datos ha resultado ser mucho mayor para el poco extra de seguridad conseguido.

Un ataque de tablas arcoiris es un método utilizado por atacantes para descifrar contraseñas hash. Este tipo de ataque aprovecha tablas precomputadas de hashes y contraseñas correspondientes para encontrar coincidencias de manera eficiente, evitando la necesidad de calcular los hashes en tiempo real.

5.2.4 Canales inseguros para la transmisión de datos sensibles.

Dicha posibilidad de vulnerabilidad, ocurre cuando se usan versiones obsoletas de TLS/SSL. El uso de versiones antiguas de TLS (1.0, 1.1) y SSL hace que la web sea vulnerable a múltiples ataques, como POODLE y BEAST. Esto expone los datos sensibles a ser interceptados o descifrados.

Solución:

El objetivo de la solución es comprobar dicha situación y arreglarlo en caso de ser necesario

En nuestro caso, en la configuración propia de nginx, tenemos especificado que las versiones a usar de TLS sean versiones modernas:

```
ssl_protocols TLSv1.2 TLSv1.3;
```

Lo cual mitiga la posibilidad de esta vulnerabilidad en gran medida.

Por otro lado, respecto a SSL, usamos claves autofirmadas, que no es lo ideal pero para un entorno de prueba (no de producción) es algo normal y útil, aunque por otro lado podríamos en un futuro ejecutar nuestra web en un dominio registrado, usando entonces claves SSL generadas y de confianza mediante cleverbot por ejemplo. Además de ello, para solventar el problema de “man in the middle”, dichas claves son montadas mediante nginx.conf desde un directorio fuera del alcance del usuario, esto es:

La aplicación web tiene disponible el acceso de los archivos dentro de /SGSSI-Proyecto-main/app , mientras que nginx.conf está guardado dentro de /SGSSI-Proyecto-main/nginx.

```
ssl_certificate /etc/nginx/certs/cert.crt;  
ssl_certificate_key /etc/nginx/certs/cert.key;  
ssl_protocols TLSv1.2 TLSv1.3;
```

5.2.5 Exposición de datos sensibles en registros o logs:

Los sistemas a menudo registran datos sensibles de error durante su desarrollo, los cuales aparecerán en los logs internos del contenedor.

Solución:

Para gestionar dicho funcionamiento y monitorearlo de forma correcta, lo que vamos a hacer es generar una carpeta para logs, dentro del Dockerfile, dándole los permisos de escritura necesarios para que salgan los logs. También cargaremos en docker-compose.yml, en la sección de la web, la carpeta en la que se cargarán dichos logs a nuestra máquina local. Además, configuraremos un custom.ini que determinará las reglas de gestión de logs:

- Paso 1:

```

; Configuraciones de errores
log_errors = On
error_log = /var/log/php_errors.log
display_errors = Off
error_reporting = E_ALL

```

```

FROM php:7.2.2-apache

# Instalar extensiones necesarias
RUN docker-php-ext-install mysqli

# Crear un directorio para logs de PHP
RUN mkdir -p /var/log && \
    chmod 777 /var/log

# Incluir un archivo de configuración personalizado
COPY conf/custom.ini /usr/local/etc/php/conf.d/custom.ini

```

La imagen de la izquierda nos muestra el archivo custom.ini:

- display_errors = Off: Esto es correcto, ya que no deseas que los errores se muestren en pantalla (por motivos de seguridad).
- log_errors = On: Esto también es correcto, ya que permite registrar errores en los logs.
- error_log = /var/log/php_errors.log: Apunta a un archivo de logs donde PHP debería estar escribiendo.
- error_reporting = E_ALL: se encarga de reportar los logs, permitiendo su escritura.

La imagen de la derecha, muestra cómo generamos la carpeta de logs y cargamos las configuraciones necesarias para los logs determinados en custom.ini.

```

version: '3'

services:
  web:
    build: ./
    environment:
      - ALLOW_OVERRIDE=true
    volumes:
      - ./app:/var/www/html/
      - ./conf/custom.ini:/usr/local/etc/php/conf.d/custom.ini
      - ./logs:/var/log/ # Carpeta local para los logs
    networks:
      - my-network
    links:
      - mariadb

```

Dentro de nuestro docker-compose.yml, también vamos a modificar “volumes” dentro del servicio “web”, cargando así la carpeta log del contenedor en el entorno local y cargando también el archivo de configuraciones conf.ini de forma correcta.

Además de estos cambios, para que se muestren errores en el log, vamos a introducir alertas tipo `error_log()` en cada forma de error para monitorear el proceso de login:

```
if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    error_log("error de validación csrf");
    die("Error de validación CSRF.");
}
```

```
if (intval($response_keys["success"]) !== 1) {
    header('Location: login.php?error=captcha_failed');
    error_log("captcha fallido");
    exit();
}
```

```
if (!is_null($user['bloqueado_hasta']) && new DateTime() < new
DateTime($user['bloqueado_hasta'])) {
    $bloqueado_hasta = new DateTime($user['bloqueado_hasta']);
    echo "Cuenta bloqueada. Intenta de nuevo después de: " . $bloqueado_hasta->format('Y-m-d
H:i:s');

    error_log("cuenta bloqueada");
    exit();
}
```

```
if (sendVerificationEmail($email, $verification_code)) {
    header('Location: verify_code.php');
    exit();
} else {
    error_log("no se pudo enviar el correo de verificación");
    echo "No se pudo enviar el correo de verificación.";
    exit();
}
```

```
} else {
    // Incrementar intentos fallidos
    $intentos_fallidos = $user['intentos_fallidos'] + 1;
    $bloqueado_hasta = null;
    error_log("Intento de inicio de sesión fallido: contraseña incorrecta.");
    if ($intentos_fallidos >= 3) {
        $bloqueado_hasta = (new DateTime())->add(new DateInterval('PT15M'))->format('Y-m-d
H:i:s');

        error_log("se procede a bloquear la cuenta");
    }
}
```


```

    header('Location: login.php?error=incorrect_password');
    error_log("contraseña incorrecta");
    exit();
  }
} else {
  // Usuario no encontrado
  header('Location: login.php?error=user_not_found');
  error_log("Intento de inicio de sesión fallido: usuario no encontrado.");
  exit();
}

```

De esta forma, hemos configurado muchos mensajes descriptivos de error, para dar una idea de lo que está pasando sin dar información delicada.

Una idea de cómo aparecen dichos logs en la carpeta dentro de nuestro entorno local:



```

Open  ~/Downloads/SGSSI-Proyecto-main/logs  php_errors.log
[22-Nov-2024 13:35:31 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:35:35 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:35:37 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:40:05 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:40:07 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:40:17 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:40:18 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:47:00 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:47:02 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:47:25 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2
[22-Nov-2024 13:47:27 UTC] Intento de inicio de sesión fallido: usuario no encontrado.
[22-Nov-2024 13:47:27 UTC] PHP Notice: session_start(): A session had already been started -
ignoring in /var/www/html/csrf.php on line 2

```

6. Inyección

6.1. Descripción

La vulnerabilidad de inyección es la tercera más común en el ámbito de las páginas web^[1], además de por su frecuencia, son especialmente peligrosas por la naturaleza de los datos que compromete (bases de datos, usuarios, contraseñas, contenido oculto de la página, métodos de pago...).

Aunque existen diversos tipos de ataques de inyección, la mayoría se llevan a cabo introduciendo líneas de código malicioso en campos de búsqueda, urls y formularios de registro o inicio de sesión.

6.2. SQL Injection

Definición

Las inyecciones SQL se ejecutan mediante la introducción de código SQL en campos de texto de una página web. Esto, si existe dicha vulnerabilidad y el ataque se ejecuta correctamente, permite el acceso a información confidencial almacenada en una base de datos de la página.

Este tipo de vulnerabilidad es causada por una mala sanitización o validación del código y por el uso de consultas dinámicas. Es decir, **no se comprueba el tipo de datos que se introducen en la consulta**, y encima **lo escrito forma la query**.

6.2.1. Análisis mediante SQLMap

SQLMap es una herramienta que permite la realización automática de ataques del tipo sql injection. Como se puede comprobar en la figura, la página web no es vulnerable a dichos ataques:

```
(aitor@Aitor)-[~]
$ sqlmap -u "http://localhost:81/login.php" --data "username=*&password=*" --level 5 --risk 3 -v 0

  ____
  |  _ \| | | | | |
  | |_) | |_| |
  |  _<| | | |
  |_| \_|_|_|_|

{1.8.9#stable}

  ____
  |  _ \| | | | | |
  | |_) | |_| |
  |  _<| | | |
  |_| \_|_|_|_|

  |V...  |  https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 17:51:49 /2024-11-10/

custom injection marker ('*') found in POST body. Do you want to process it? [Y/n/q] y
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=d76bf5e78e7...7fd422ed9d'). Do you want to use those [Y/n] y
it is recommended to process only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] y
[17:52:40] [CRITICAL] all tested parameters do not appear to be injectable. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'
```

Comando utilizado: [2]

```
$ sqlmap -u 'http://localhost' --data 'username=*&password=*' --level 5 --risk 3 -v 0
```

-u url a atacar

--data los datos que inyectará sqlmap con el método POST.

–**level** la cantidad de métodos a comprobar [1-5].

--risk la agresividad del ataque [1-3].

6.2.2. Prevención de la vulnerabilidad ¿Porqué nuestra página ya es segura?

Para solucionar los problemas descritos en el apartado de definición utilizamos dos estrategias:

1. Validación y sanitización de los inputs. (Aseguramos que el formato es correcto)
2. Uso de declaraciones preparadas y queries parametrizadas. (Evitamos que el input altere la query)

De la primera medida de prevención (solo validación) se encarga el archivo *"SGSSI-Proyecto/app/validaciones.js"*. Mientras que la segunda ya la implementamos en la fase de desarrollo de la página, como se puede observar en la figura:

```
// Consulta para verificar usuario
$stmt = $conn->prepare("SELECT * FROM usuarios WHERE usuario = ?");
$stmt->bind_param("s", $username);
$stmt->execute();
$result = $stmt->get_result();
```

6.3. XSS (Cross-Site Scripting)

Definición

Los ataques de XSS son un tipo de inyección, mediante la cual el usuario atacante prepara código malicioso en una página web que será ejecutado por otros usuarios. A diferencia de las inyecciones SQL, que se dan en consultas a bases de datos, las vulnerabilidades XSS surgen de contenido añadido por los usuarios a la página web como comentarios, posts, fotos...

Una vez inyectado el código, el navegador de los usuarios que accedan a la página web ejecutarán dicho código, pues este se fía de la página en sí y no es capaz de reconocer un script legítimo de uno malicioso. Por ello, el foco en cuanto a seguridad ha de ponerse en evitar la inyección de dicho código. [\[3\]](#)

6.3.1. Análisis de la vulnerabilidad en “items.php”

Como bien hemos mencionado en el apartado [6.2.2](#), las dos mejores estrategias para prevenir un ataque de inyección son la validación del input y el uso de declaraciones preparadas. En el caso de las páginas de login y registro (process_login.php y process_register) hemos implementado ambas medidas. Sin embargo, aunque hayamos implementado el uso de declaraciones preparadas y queries parametrizadas en 'process_add_item.php' y 'process_editar_item.php', no sanitizamos los inputs de los usuarios.

Teniendo esto en cuenta nos encontramos ante un ambiente perfecto para hallar una vulnerabilidad XSS; campos de texto para añadir contenido a una página web y la no sanitización de dicho texto.

6.3.2. Simulación de ataque XSS

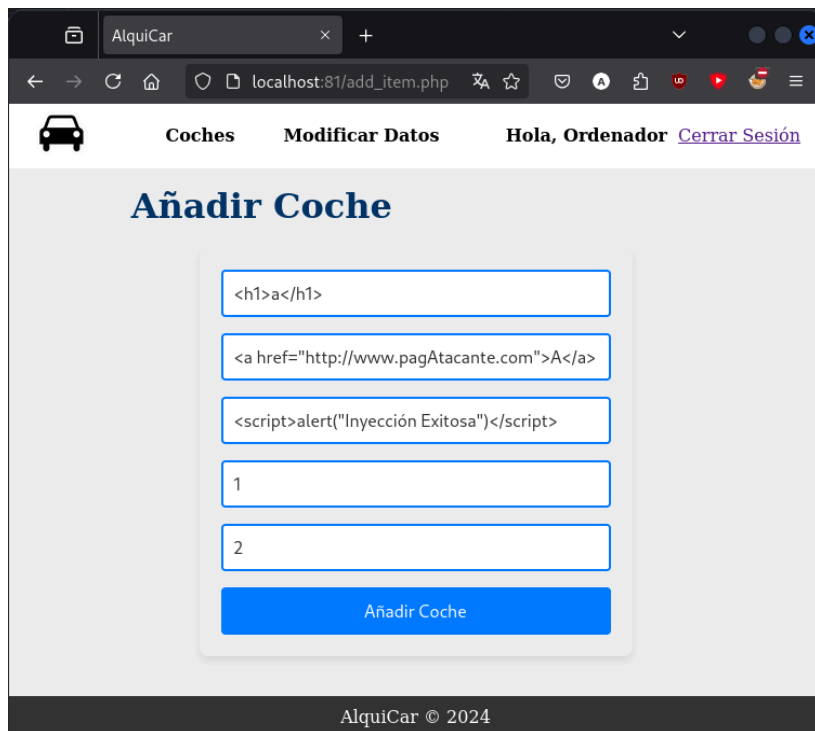
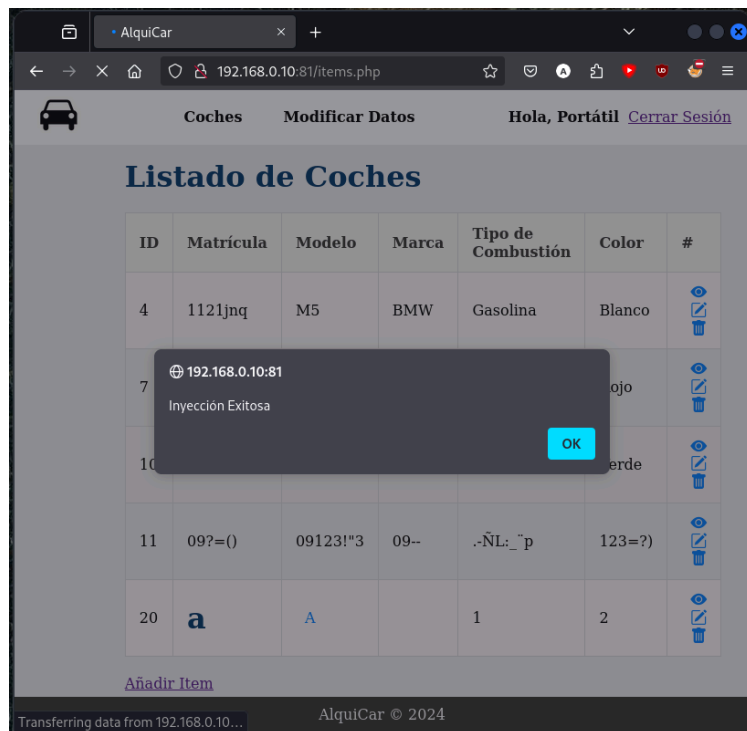
Para demostrar el funcionamiento de la vulnerabilidad voy a simular un ataque de XSS desde mi ordenador al portátil (hosteando la página en el ordenador y conectándome desde el portátil mediante port forwarding). En este ejemplo el atacante será el usuario del ordenador y la víctima la del portátil (se diferencian por el usuario y la url).

El primer paso consiste en comprobar que el texto no se valide, esto se hace creando un coche con campos sin sentido, llenos de símbolos y números donde no tendría que haberlos.

Listado de Coches

ID	Matrícula	Modelo	Marca	Tipo de Combustión	Color
4	1121jmq	M5	BMW	Gasolina	Blanco
7	7302-yaq	Seat	BMW	Gasolina	Rojo
10	0000-AAA	Seat	Honda	Híbrido	Verde
11	09?=(09123!"3	09--	.-ÑL: _"p	123=?)

Después inyectamos el código deseado, en este ejemplo: un header html, un link a la página web del atacante y un script que lanza una alerta con el mensaje '*Inyección Exitosa*'.

ID	Matrícula	Modelo	Marca	Tipo de Combustión	Color	#
4	1121jmq	M5	BMW	Gasolina	Blanco	
7					rojo	
10					verde	
11	09?=(09123!"3	09--	.-ÑL:_"p	123=?)	
20	a	A		1	2	

Con esta simple simulación se demuestra lo fácil que es atacar una página web con tan solo un par de líneas de código, teniendo una infinidad de posibilidades que explotar. Desde inyección de código HTML, php, javascript, redirects a páginas de fishing, descargar malware, robo de cookies de sesión... [4]

6.3.3. Solución

Para solucionar esta vulnerabilidad la estrategia más sencilla consiste en la sanitización de los inputs. Para ello pasamos los datos a strings y utilizamos expresiones regulares para convertir los inputs en strings alfanuméricos antes de introducirlos a la base de datos.

```
if(isset($_POST['item_add_submit'])) {
    $matricula = $_POST['matricula'];
    $modelo = $_POST['modelo'];
    $marca = $_POST['marca'];
    $tipo_combustion = $_POST['tipo_combustion'];
    $color = $_POST['color'];

    // Recive y sanitiza los datos del formulario.
    $matricula = preg_replace('/^[^a-zA-Z0-9-]/', '', (string)$POST['matricula']);
    // matricula tiene un regex distinto para permitir -.
    $modelo = preg_replace('/^[^a-zA-Z0-9]/', '', (string)$POST['modelo']);
    $marca = preg_replace('/^[^a-zA-Z0-9]/', '', (string)$POST['marca']);
    $tipo_combustion = preg_replace('/^[^a-zA-Z0-9]/', '', (string)$POST['tipo_combustion']);
    $color = preg_replace('/^[^a-zA-Z0-9]/', '', (string)$POST['color']);
}
```

28	1123-AGV	ahrefhttpwww ejemplocmAa	scriptalertSanitizacinscript	h1Gasolinah1	Verdea	  
----	----------	--------------------------	------------------------------	--------------	--------	---

7. Configuración y diseño inseguro

7.1. Descripción

El diseño inseguro es cuando se crean sistemas o aplicaciones que no consideran adecuadamente los principios de seguridad desde que se ha creado. Los principales diseños inseguros se dan en los controles de acceso o la gestión segura de los datos. La falta de análisis de riesgos y pruebas de seguridad dejan expuesto al proyecto a ataques.

7.2. Posibles causas

7.2.1. Imprimir mensajes de error que contengan información confidencial

Cuando programamos un sistema pensamos en los posibles errores que se pueden producir. En esos errores que se pueden producir solemos poner un print que nos informa de que se trata. Hay veces que utilizamos herramientas externas como puede ser una base de datos. Cuando hacemos una petición sql, se pueden dar errores. Cuando se dan esos errores si los imprimimos, cualquier persona podría aprovecharse y atacarlos.

Solución:

Para solucionar este error sustituimos lo que pueda decir la base de datos por nuestros propios comentarios como:

```
<p style="color: red;">
  <?php
    switch ($_GET['error']) {
      case 'captcha_failed':
        echo "Falló el CAPTCHA. Por favor, inténtalo de nuevo.";
        break;
      case 'incorrect_password':
        echo "Contraseña incorrecta. Inténtalo nuevamente.";
        break;
      case 'user_not_found':
        echo "Usuario no encontrado. Verifica tu nombre de usuario.";
        break;
      default:
        echo "Ocurrió un error. Inténtalo nuevamente.";
        break;
    }
  ?>
</p>
```

En el siguiente código podemos ver como imprimamos el error que daba:

```
// Ejecuta la consulta y verifica si fue exitosa
if ($conn->query($sql) === TRUE) {
    header('Location: /items.php');
    exit;
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
```

Ahora hemos editado el código para que imprima el error que imprima una información reducida pero suficiente para que podamos identificarla.

```
// Ejecuta la consulta y verifica si fue exitosa
if ($conn->query($sql) === TRUE) {
    header('Location: /items.php');
    exit;
} else {
    echo "Error: No se ha podido añadir el item. <br>" . $sql . "<br>";
}
```

7.2.2. Almacenamiento de credenciales sin protección

Causa:

El almacenamiento inseguro de credenciales se refiere a la práctica de guardar información sensible, como contraseñas y claves de API, directamente en archivos de configuración que podrían ser accesibles por personas no autorizadas. Esto puede dar lugar a fugas de información que comprometan la seguridad del sistema.

Solución Implementada:

Se han realizado algunas mejoras importantes para manejar de manera segura las credenciales en el proyecto:

1. Docker Compose (docker-compose.yml):

- **Credenciales de Base de Datos:** Aún se observan credenciales de la base de datos (MYSQL_ROOT_PASSWORD, MYSQL_DATABASE, MYSQL_USER, MYSQL_PASSWORD) incluidas directamente en el archivo docker-compose.yml. Esta práctica representa un riesgo si el archivo se comparte públicamente o es accesible por terceros. Por lo que utilizaremos un archivo .env para almacenar las variables.

```

mariadb:
  image: mariadb:latest
  container_name: mariadb # Nombre fijo para el cont
  environment:
    - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
    - MYSQL_DATABASE=${MYSQL_DATABASE}
    - MYSQL_USER=${MYSQL_USER}
    - MYSQL_PASSWORD=${MYSQL_PASSWORD}
  ports:
    - "3307:3306"
  volumes:
    - mysql-data:/var/lib/mysql
  networks:
    - my-network
  
```

2. Mejora Implementada:

- **Archivo .env:** Para asegurar un almacenamiento seguro, se movieron las credenciales a un archivo .env. (Este archivo no se debería de publicar, aunque se publicará en el github) De este modo, se puede evitar que la información sensible se incluya directamente en los archivos del repositorio. Aquí se muestra cómo usar este archivo en docker-compose.yml:

- Ejemplo de configuración mejorada con .env:

```

1 MYSQL_ROOT_PASSWORD=rootpassword
2 MYSQL_DATABASE=mydatabase
3 MYSQL_USER=myuser
4 MYSQL_PASSWORD=mypassword
5
  
```

7.2.3. Violación de los límites de confianza

Causa: Una violación de los límites de confianza se refiere a la exposición innecesaria de servicios internos a usuarios o procesos externos. Esto puede suceder si no se restringen adecuadamente las conexiones entre los módulos del sistema, permitiendo que partes de la infraestructura interactúen de forma no segura o innecesaria. Esta falta de control puede generar vulnerabilidades explotables por un atacante para obtener acceso a recursos internos sensibles.

Solución Implementada:

Basado en las configuraciones ya presentes en el archivo `docker-compose.yml`, se han tomado ciertas medidas que ayudan a proteger y definir claramente los límites de confianza dentro del proyecto:

1. Redes Aisladas con Docker (`docker-compose.yml`):

- Todos los servicios (`web`, `mariadb`, `phpmyadmin`, `nginx`) están definidos en una red común llamada `my-network` con el driver `bridge`. Esta configuración crea una red aislada, dentro de la cual los contenedores pueden comunicarse entre sí sin estar accesibles desde la red pública a menos que los puertos se exponen explícitamente.

```
53 networks:  
54   my-network:  
55     driver: bridge  
56
```

2. Beneficio de la Red Aislada:

- Utilizar `bridge` como driver para la red garantiza que los servicios internos (como `mariadb`) no estén accesibles desde fuera de la red Docker a menos que los puertos sean expuestos manualmente. Esto limita la comunicación entre servicios únicamente a aquellos que están en la misma red definida, evitando así accesos no autorizados desde fuera del entorno Docker.

3. Exposición de Puertos Controlada:

- La exposición de puertos está claramente definida, lo cual permite mantener un control sobre qué servicios están accesibles desde fuera del entorno Docker. El servicio `nginx` está configurado para exponer los puertos 443 (HTTPS) y 81 (HTTP redirigido a HTTPS), mientras que `phpmyadmin` y `mariadb` no exponen directamente sus puertos a la red pública, lo cual limita el acceso externo a estos servicios.

```
1 #bloque para la web  
2 server {  
3     listen 443 ssl;  
4     server_name web.localhost;  
5
```

4. Beneficio de la Exposición Controlada de Puertos:

- Mantener solo los puertos necesarios abiertos y expuestos asegura que los servicios internos, como la base de datos (`mariadb`) y `phpmyadmin`, no puedan ser accedidos directamente desde el exterior. Esto reduce el riesgo

de ataques directos a estos servicios, protegiendo así la infraestructura de posibles vulnerabilidades externas.

Conclusión:

La configuración de redes en Docker y la exposición controlada de puertos son elementos ya presentes en el proyecto que ayudan a prevenir la violación de los límites de confianza. Al definir una red específica (my-network) y limitar la exposición de los puertos, se establece una separación clara entre los servicios internos y externos, asegurando que solo los componentes necesarios puedan comunicarse entre sí. Estas medidas permiten proteger los recursos internos del sistema y reducir la superficie de ataque, garantizando que cada módulo solo tenga acceso a lo estrictamente necesario.

8. Componentes vulnerables y obsoletos

8.1. Descripción

En esta sección vamos a ver las vulnerabilidades correspondientes a “Componentes vulnerables y obsoletos”. Estas vulnerabilidades se dan cuando no cuenta con soporte o está desactualizado. Esto incluye el sistema operativo, el servidor web o de aplicaciones, el sistema de administración de bases de datos, las aplicaciones, las API y todos los componentes, los entornos de ejecución y las bibliotecas.

Si no escanea periódicamente en busca de vulnerabilidades y no se suscriben a boletines de seguridad relacionados con los componentes que utiliza.

8.2. Cómo prevenir

Para prevenir este tipo de ataques la OWASP nos dice algunas cosas que tenemos que prevenir en nuestro proyecto.

- Eliminar dependencias no utilizadas, componentes, archivos y documentos.
- Realizar un inventario continuo de las versiones de los componentes del lado del cliente y del lado del servidor y sus dependencias utilizando herramientas para revisar las vulnerabilidades.
- Obtener componentes de fuentes oficiales.

8.2. Posibles causas

8.2.1. Vulnerabilidades de PHPMailer

PHPMailer es una librería que utilizamos en nuestro proyecto para enviar correos electrónicos. Como el envío de correos lo utilizamos para mejorar la seguridad tendremos que mejorar la seguridad. Por ello, tenemos que enviar los correos seguros (TLS o SSL) para ellos hemos añadido mejoras en el código como las siguientes líneas.

```
$mail->SMTPAuth = true;  
$mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
```


9. Fallos de identificación y autenticación

9.1. Descripción

Esta categoría engloba errores relacionados con la gestión de la autenticación (cómo se verifica la identidad de un usuario) y la identificación (cómo se determina quién es el usuario). Los fallos en esta área pueden permitir accesos no autorizados o un uso indebido de la identidad de usuarios legítimos.

9.2. Posibles causas

9.2.1. Ausencia o mal diseño de autenticación:

Se debe al hecho de permitir el acceso sin requerir autenticación o con mecanismos débiles.







Ejemplo:

- Funciones sensibles accesibles sin iniciar sesión.
- URLs previsibles que no verifican permisos.

Solución:

En nuestra web, para visualizar los coches no hace falta iniciar sesión, pero a la hora de querer eliminar un coche o editarlo, se tendrá que iniciar sesión con un usuario dado de alta.

Un ejemplo de la gestión de coches si no se está logueado:

Coches							Login
							
Listado de Coches							
ID	Matrícula	Modelo	Marca	Tipo de Combustión	Color	#	
2	1121-jns	M5	BMW	Gasolina	negro		
4	1121jmq	M5	BMW	Gasolina	Blanco		
6	1821-jns	M5	BMW	Gasolina	Rojo		
7	7302-yaq	Seat	BMW	Gasolina	Rojo		
9	2121-aaa	Seat	BMW	Gasolina	Rojo		

Y la misma sección estando logueado:



Coches
 [Modificar Datos](#)

Hola, diegogar1
 [Cerrar Sesión](#)

Listado de Coches

ID	Matricula	Modelo	Marca	Tipo de Combustión	Color	#
2	1121-jns	M5	BMW	Gasolina	negro	  
4	1121jmq	M5	BMW	Gasolina	Blanco	  
6	1821-jns	M5	BMW	Gasolina	Rojo	  
7	7302-yaq	Seat	BMW	Gasolina	Rojo	  
9	2121-aaa	Seat	BMW	Gasolina	Rojo	  

Por ello, podemos decir que esta causa posible, está bien definida y es evitada.

9.2.2. Contraseñas débiles o políticas de contraseñas inadecuadas:

Permitir contraseñas fáciles de adivinar o no implementar controles adecuados.

Ejemplo:

- Contraseñas como "123456" o "password".
- No exigir el cambio periódico de contraseñas en sistemas críticos.

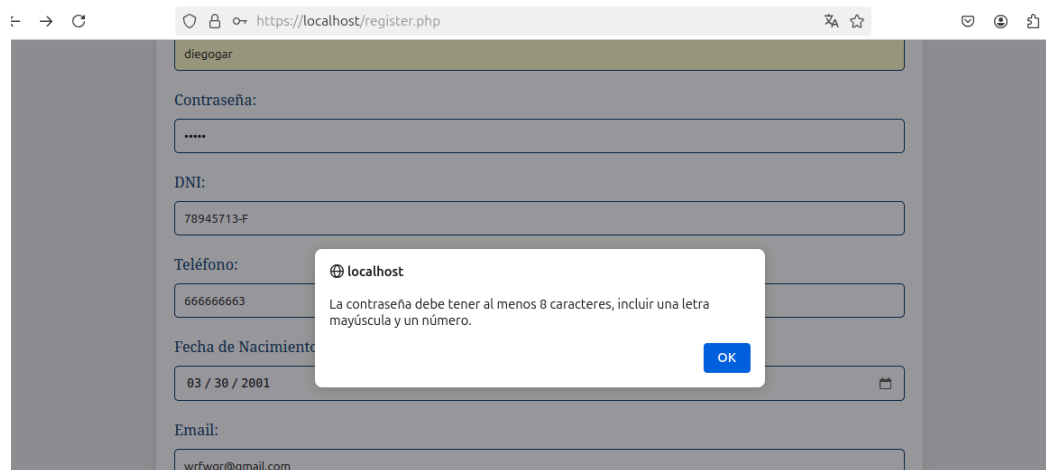
Solución:

Para evitar la posibilidad de incluir contraseñas débiles, podríamos hacer algún cambio en la hoja de registro, haciendo que en el campo de contraseña se requieran al menos 8 dígitos, 1 número y una mayúscula (por ejemplo), para ello editaremos el archivo validaciones.js añadiendo lo siguiente:

```
const password = document.getElementById("password").value;
```

```
// Validar contraseña
const passwordPattern = /^(?=.*[A-Z])(?=.*\d){8,}$/;
if (!passwordPattern.test(password)) {
  alert("La contraseña debe tener al menos 8 caracteres, incluir una letra mayúscula y un número.");
  return false;
}
```

De esta forma nos aseguramos una estructura firme y fuerte en la contraseña del usuario. Probando su funcionalidad, si introducimos una contraseña diferente al estipulado en el formato, nos saldrá lo siguiente:



9.2.3. Falta de autenticación multifactor (MFA):

Confiar únicamente en contraseñas, lo cual es insuficiente para proteger cuentas en sistemas de alta sensibilidad.

Ejemplo:

- Un atacante roba la contraseña y puede acceder sin restricciones debido a la falta de un segundo factor (como un código enviado al teléfono).

Solución:

En este caso, la solución implica descargar la carpeta PHPMailer del repositorio de github oficial, y de esta forma gestionar el inicio de sesión para que envíe un email con el código al gmail seleccionado en los datos del usuario que pretende iniciar sesión, de esta forma aumentamos la seguridad.

Para ello, crearemos dos nuevos archivos .php dentro de la carpeta app: send_email.php(que gestionará los documentos .php necesarios para enviar un gmail mediante smtp) y verify_code.php (que procesa el código de respuesta y lo comparará para el correcto inicio de sesión). Además, modificaremos el archivo process_login.php para ajustarse a las nuevas medidas.

Nueva página de verificación de código:



Mensaje recibido en el correo del usuario:



En este caso tenemos configurado en el archivo `send_email.php` el correo diego1314gc@gmail.com y el cuerpo del mensaje de esa forma, pero se podría elegir cualquier otro gmail siempre y cuando se genere una clave única de aplicación en ajustes de la propia cuenta:

```

Open  send_email.php
~/Downloads/SGSSI-Proyecto-entrega_1/app

register.php  csrf.php  cleaned_index.p  validaciones.js  process_register  edited_process_  process_login.pl  v

<?php
require 'PHPMailer/src/PHPMailer.php';
require 'PHPMailer/src/Exception.php';
require 'PHPMailer/src/SMTP.php';

use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;

function sendVerificationEmail($email, $code) {
    $mail = new PHPMailer(true);

    try {
        // Configuración del servidor SMTP
        $mail->isSMTP();
        $mail->Host = 'smtp.gmail.com'; // Cambia esto si usas otro proveedor
        $mail->SMTPAuth = true;
        $mail->Username = 'diego1314gc@gmail.com'; // Tu correo electrónico
        $mail->Password = 'gqpo bvdz jxua yzez'; // Contraseña o clave de aplicación
        $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
        $mail->Port = 587;

        // Configuración del correo
        $mail->setFrom('diego1314gc@gmail.com', 'Verificacion');
        $mail->addAddress($email);

        $mail->isHTML(true);
    }
}

```

¿Cómo funciona el inicio de sesión mediante verificación?

Habiendo explicado para qué sirven los dos nuevos archivos `send_email.php` y `verify_code.php`, nos queda explicar cómo redirige el archivo `process_login.php` a estos archivos:

```
// Verificación de contraseña
if (password_verify($password, $user['password'])) {
    $email = $user['mail']; // Extraer el correo del usuario

    // Generar un código de verificación
    $verification_code = rand(100000, 999999);
    $_SESSION['verification_code'] = $verification_code;
    $_SESSION['user_email'] = $email;
    $_SESSION['temp_username'] = $user['usuario'];

    // Enviar el código al correo del usuario
    if (sendVerificationEmail($email, $verification_code)) {
        header('Location: verify_code.php'); // Redirigir a la página de verificación
        exit();
    } else {
        echo "No se pudo enviar el correo de verificación.";
    }
} else {
    // Contraseña incorrecta
    header('Location: login.php?error=incorrect_password');
    exit();
}
} else {
    // Usuario no encontrado
    header('Location: login.php?error=user_not_found');
    exit();
}
}
```

De esta manera, creamos un código de verificación mediante `$verification_code = rand(100000,999999)`, que se nos almacena en `$_SESSION['verification_code']`, que más tarde enviaremos a la función `sendVerificationCode` dentro de `send_email`, que en caso de que sean iguales, `process_login.php` redirigirá el inicio a `verify_code.php`:

```
<?php
session_start();
require_once 'con.php'; // Asegúrate de incluir la conexión si necesitas más datos del usuario

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $input_code = $_POST['code'];

    if ($input_code === $_SESSION['verification_code']) {
        // Código correcto, almacena datos importantes en la sesión
        $_SESSION['logged_in'] = true;
        $_SESSION['username'] = $_SESSION['temp_username']; // Asume que 'temp_username' fue almacenado durante el proceso de login
        $_SESSION['email'] = $_SESSION['user_email']; // Usa el correo del usuario almacenado previamente

        // Limpia los datos temporales
        unset($_SESSION['verification_code']);
        unset($_SESSION['temp_username']);

        // Redirige a la página principal
        header('Location: index.php');
        exit();
    } else {
        // Si el código es incorrecto, muestra un mensaje de error
        echo "El código ingresado es incorrecto.";
    }
}
?>
```

De esta forma, si el código introducido por el usuario en el formulario concuerda, mediante header(“Location: index.php”) nos redirigirá a la página principal con la sesión ya iniciada correctamente.

9.2.4. Mal manejo de tokens de sesión:

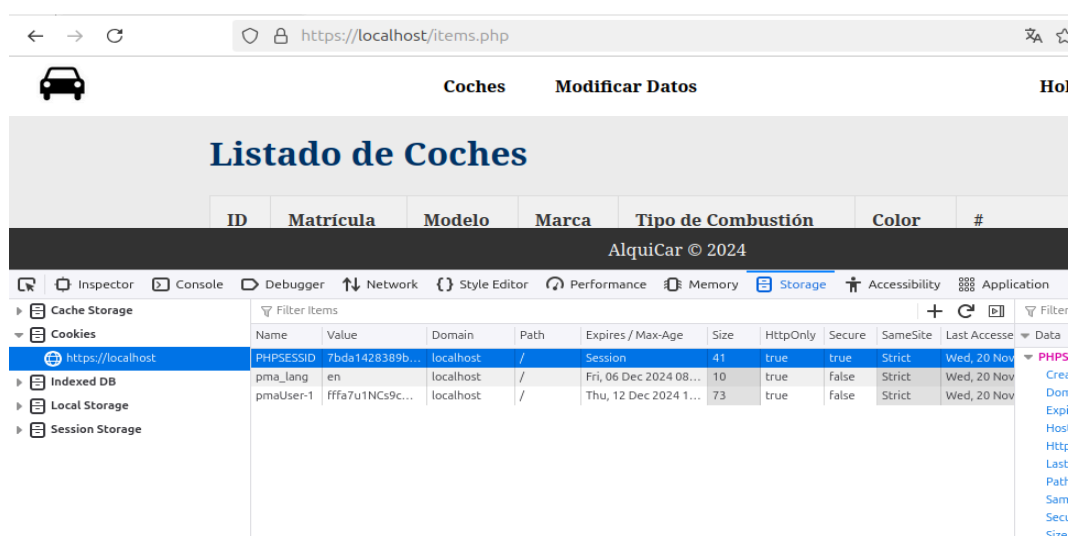
La gestión inadecuada de tokens permite que un atacante los robe y use para acceder como si fuera el usuario legítimo.

Ejemplo:

- Falta de cifrado: Tokens enviados en texto plano (HTTP en lugar de HTTPS).
- Reutilización: Tokens que no caducan tras cerrar sesión.

Solución:

En nuestro caso, gracias a la configuración previa con nginx, podemos apreciar que la cookie que envía las contraseñas y datos importantes, tiene una duración de la sesión, por tanto una vez cerrada esta, se eliminará:



9.2.5. Falta de protección contra ataques CSRF:

La aplicación no valida que las solicitudes provengan del usuario legítimo.

Ejemplo:

- Un atacante utiliza una cookie de sesión activa para enviar solicitudes en nombre del usuario.
- Solución común: Uso de tokens Anti-CSRF para verificar la legitimidad de la solicitud.

Solución:

En este caso, podemos ver el flag indicado en el reporte ZAP: ***Absence of Anti-CSRF Tokens:***

La ausencia de tokens Anti-CSRF (Cross-Site Request Forgery) es una vulnerabilidad que deja a las aplicaciones web expuestas a ataques de tipo CSRF. Estos ataques ocurren cuando un atacante engaña a un usuario autenticado para que ejecute acciones no deseadas en una aplicación web en la que tiene privilegios.

Para solucionar esto, gestionamos el token CSRF a través del archivo csrf.php que hemos creado y está integrado en los formularios de los archivos login.php, process_login.php, register.php y process_register.php.

```

<?php
session_start();

// Generar el token CSRF si no existe
function generate_csrf_token() {
    if (empty($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
    }
}

// Obtener el token CSRF
function get_csrf_token() {
    generate_csrf_token();
    return $_SESSION['csrf_token'];
}

// Validar el token CSRF
function validate_csrf_token($token) {
    if (empty($token) || $token !== $_SESSION['csrf_token']) {
        die("CSRF token inválido.");
    }
}
?>

```

Aquí tenemos el código referente a csrf.php, el cual genera el token mediante generate_csrf_token() y es retornado por la función get_csrf_token().

En los formularios de login.php y register.php, el token se inserta como un campo oculto:

```

<!-- Campo oculto para el token CSRF -->

<button type="submit" class="login-button" id="login_submit">Iniciar Sesión</button>
<br><br>

```

El token se envía junto al resto de los parámetros requeridos por el formulario, antes del botón de “submit” o enviar.

En los archivos process_login.php y process_register.php, el token se valida al recibir el formulario:

```

validate_csrf_token($_POST['csrf_token']);

if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    die("Error de validación CSRF.");
}

```

Si los tokens no coinciden o no están presentes, se detiene la ejecución con el mensaje de error “Error de validación CSRF”.

9.2.6. Falta de bloqueo tras intentos fallidos de autenticación:

No limitar los intentos de inicio de sesión permite ataques de fuerza bruta.

Ejemplo:

- Un atacante prueba miles de combinaciones de contraseñas sin ser bloqueado.

Solución:

Para subsanar esta posibilidad, vamos a implementar un segmento de código en process_login.php que limite los intentos de sesión por usuario a 5, de esta forma, tras el quinto intento fallido, saldrá este mensaje:



Además, para que esto sea necesario, hemos tenido que añadir dos columnas nuevas a la tabla usuarios.

```

ALTER TABLE usuarios ADD COLUMN intentos_fallidos INT DEFAULT 0;
ALTER TABLE usuarios ADD COLUMN bloqueado_hasta DATETIME NULL;

```

Estos cambios lograrán el cometido de evitar que se hagan intentos ilimitados para acceder a una cuenta.

10. Fallos en la integridad de datos y software

10.1. Descripción

En esta sección vamos a identificar los fallos de seguridad de “Integridad de datos y software” que tiene nuestro proyecto. Los fallos de integridad de datos y software son aquellos en los que no se garantiza que los datos y software puedan ser alterados de forma autorizada. Estos problemas pueden hacer que no se garantice la integridad de la información o el comportamiento del software. Esto puede producir riesgos en la seguridad y en el funcionamiento de nuestro proyecto.

10.2. Cómo prevenirlos

Para prevenir este tipo de ataques la OWASP nos dice algunas cosas que tenemos que prevenir en nuestro proyecto.

- Utilizar firmas digitales o mecanismos similares para verificar que el software o los datos vienen de la fuente esperada y no hayan sido manipulados.
- Bibliotecas y dependencias como npm o Maven, utilicen repositorios de confianza.
- Utilizar herramientas de seguridad para verificar que los componentes no contienen vulnerabilidades conocidas.
- Seguimiento de los cambios del código y configuración para minimizar la posibilidad de que se introduzcan códigos o configuraciones maliciosas.

10.3. Posibles causas

10.3.1. Manipulación de archivos críticos del sistema

Uno de los posibles archivos críticos de configuración críticos en nuestro proyecto es **nginx.conf** que es el que maneja las solicitudes, dirige el tráfico, gestiona la seguridad, y establece parámetros clave para el funcionamiento del proyecto.

Si en un ataque modifican el fichero pueden desviar el tráfico hacia un sitio malicioso.

Solución:

Configurar permisos adecuados para los archivos sensibles y usar herramientas para detectar cambios en ellos. Como que todos los cambios que se realicen se registren en un log change.

Implementación con Linux:

Proteger los archivos críticos mediante permisos restrictivos:

```
chown root:root /etc/nginx/nginx.conf
```

```
chmod 600 /etc/nginx/nginx.conf
```

Resultado esperado:

Solo el usuario root puede modificar el archivo **nginx.conf**, previniendo manipulaciones no autorizadas.

10.3.2. Falta de verificación de integridad en las comunicaciones

Otro de los fallos más comunes que se pueden llevar a cabo en nuestro proyecto es en la comunicación Cliente - Servidor. Por ello, hemos protegido la web lo que permitirá la información durante el tránsito.

Solución:

Usar HTTPS/TLS para asegurar la comunicación y verificar que los certificados sean válidos.

Implementación en Nginx:

Configuración básica para usar HTTPS/TLS:

```
server {  
  
    listen 443 ssl;  
  
    server_name example.com;  
  
  
    ssl_certificate /etc/nginx/ssl/example.com.crt;  
    ssl_certificate_key /etc/nginx/ssl/example.com.key;  
  
    ssl_protocols TLSv1.2 TLSv1.3;  
  
}
```

Resultado esperado:

La comunicación entre el cliente y el servidor está cifrada y protegida contra manipulaciones.

10.4. Conclusión

Garantizar la integridad de los datos y del software implica cifrar la información, proteger archivos críticos del sistema, verificar la integridad del software antes de instalarlo y asegurar las comunicaciones entre módulos del sistema. Implementar estas medidas contribuye a prevenir manipulaciones no autorizadas y a mantener la seguridad y confiabilidad del sistema.

11. Fallos en la monitorización de la seguridad

11.1. Descripción

En esta sección vamos a identificar los fallos de seguridad de “Monitorización de la seguridad” que tiene nuestro proyecto. Los fallos de monitorización de seguridad son aquellos en los que no registran los cambios que se han hecho, quien los ha hecho y si han sido exitosa o denegada. Mediante este registro podemos supervisar cambios no autorizados, accesos indebidos y otros eventos críticos en tiempo real.

11.2. Cómo prevenirlos

Para prevenir este tipo de ataques la OWASP nos dice algunas cosas que tenemos que prevenir en nuestro proyecto.

- Los eventos que hay que registrar son inicio de sesión favorables o fallidos, transiciones de alto valor, ...
- Las advertencias y los errores no generan mensajes de registro, estos son inadecuados o poco claros.
- Los registros de aplicaciones y API no se monitorean para detectar actividades sospechosas.

11.3. Posibles causas

11.3.1. Ausencia de supervisión de cambios sensibles

Como hemos visto en la sección anterior uno de los archivos sensibles son los de NGINX. En el archivo **nginx.conf** no hay herramientas o mecanismos para supervisar modificaciones en archivos críticos.

Si un atacante edita el archivo **nginx.conf** para redirigir tráfico a un servidor malicioso sin ser detectado.

Solución:

Implementar un sistema de monitorización de integridad que detecte cambios no autorizados en archivos sensibles.

Implementación:

Creemos un archivo PHP **monitoring_nginx.php** que compara un hash original con el hash actual del archivo:

```
<?php

// Ruta del archivo a monitorear

$filePath = "../nginx/nginx.conf";

// Archivo donde se almacenará el hash inicial

$hashFile = "referencia.txt";

// Función para calcular el hash del archivo

function calcularHash($filePath) {

    if (!file_exists($filePath)) {

        return null;

    }

    return hash_file("sha256", $filePath);

}

// Verificar si existe un hash de referencia

if (!file_exists($hashFile)) {

    // Calcular y guardar el hash inicial

    $hash = calcularHash($filePath);

    if ($hash !== null) {

        file_put_contents($hashFile, $hash);

        echo "Hash inicial generado y almacenado.\n";

    } else {

        echo "El archivo a monitorear no existe.\n";

    }

}
```

```
}  
  
exit;  
  
}  
  
// Leer el hash almacenado y calcular el actual  
$hashReferencia = file_get_contents($hashFile);  
$hashActual = calcularHash($filePath);  
if ($hashActual === null) {  
    echo "El archivo a monitorear no existe.\n";  
    exit;  
}  
  
// Comparar los hashes  
if ($hashReferencia === $hashActual) {  
    echo "No se detectaron cambios en el archivo.\n";  
} else {  
    echo "¡Se detectaron cambios en el archivo!\n";  
    // Actualizar el hash de referencia si se desea  
    file_put_contents($hashFile, $hashActual);  
    echo "El hash de referencia se ha actualizado.\n";  
}  
  
?>
```

Resultado esperado:

Cualquier cambio en el archivo **nginx.conf** se registrará en el archivo **nginx.log**.

11.4. Conclusión

Este tipo de monitorización de seguridad nos ayuda a supervisar los cambios en los archivos críticos, y registrar los accesos a recursos sensibles generando alertas.

12. Referencias

- [1] Van der Stock, A., Glas, B., Smithline, N., & Gigler, T. (2021). OWASP Top10 2021. <https://owasp.org/Top10/>
- [2] Stampar, M., & Damele, B. (s.f.). Documentación oficial SQLMap. <https://github.com/sqlmapproject/sqlmap/wiki/Usage>
- [3] Kirsten, S. (s.f.). Cross-Site Scripting (XSS). <https://owasp.org/www-community/attacks/xss/>
- [4] Laprovittera, C. (2024). XSS Attack. <https://achirou.com/xss-attack/>