Sistemas de Gestión de Seguridad de Sistemas de Información

Sistema Web MusicCloud

Autores:
Aitana Olivera Garcia
Asier Reinoso Villalba
Andoni Garcia Beaskoetxea
Sofia Granja de Zuasnabar
Naia Ladron David
Mateo Ortiz Arnodo

Índice

1. INTRODUCCIÓN

En este documento se describe el análisis de seguridad del sistema web *MusicCloud*. Previamente diseñado para la primera entrega, sin conocimientos específicos en seguridad informática.

El objetivo es estudiar el sistema web con los conocimientos adquiridos en clases teóricas, para identificar y subsanar errores o problemas que el sistema contenga. Con el fin de conseguir una sistema web más seguro, poner en práctica la teoría y llegar a diferentes conclusiones.

Para ello se llevó a cabo una auditoría utilizando una herramienta de *pentesting* (pruebas de penetración) de código abierto. Esta herramienta, mediante escaneos automáticos, identifico vulnerabilidades en el sistema Web. Y a partir de este informe se empezó a trabajar.

URL de Github del proyecto auditado:
https://github.com/aitanaog/docker-lamp/tree/entrega_2

2. PRIMERA AUDITORÍA

La primera auditoría fue realizada tomando como base el sistema tal y como fue entregado.

Para llevar a cabo este análisis, se utilizó ZAP. Esta herramienta actúa como un proxy, interceptando todas las solicitudes y respuestas que se generan en el sitio web. De esta manera, permite identificar posibles agujeros de seguridad al examinar tanto las peticiones como las respuestas.

Al ejecutarla en el sitio web, fueron identificados un total de 15 errores. Entre las alertas más destacadas se encontraban problemas relacionados con inyección, que requieren atención prioritaria.

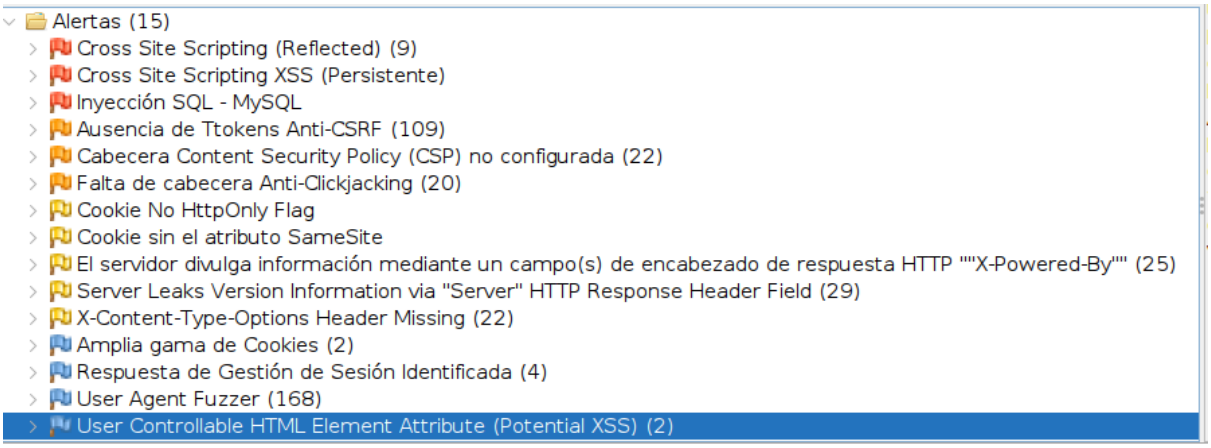

Figura 1. Errores de la primera auditoría

3. VULNERABILIDADES

3.1 ROTURA DE CONTROL DE ACCESO

La rotura del control de acceso ocurre cuando un sistema permite a los usuarios realizar acciones o acceder a recursos que deberían estar restringidos según sus roles o niveles de autorización.

En la entrega anterior, ya se usó la variable de sesión **$_SESSION[]** para asegurar que el usuario solo pueda acceder y modificar sus propios datos.

Usar **$_SESSION[]** asegura que la consulta SQL obtenga solo los datos del usuario que inició sesión. Esto previene que un atacante acceda a los datos de otros usuarios, ya que no puede manipular directamente los parámetros para consultar otra información.

```php
// Obtener el email del usuario de la sesión
$user_email = $_SESSION['user_email'];
```

```php
// CONSULTA1: obtener el ID del usuario por su email
$sql = "SELECT id FROM usuarios WHERE email = ?";
$stmt = $conn->prepare($sql);
if ($stmt === false) {
    die("Error al procesar la solicitud " . $conn->error);
}

// Vincular el parámetro y ejecutar la consulta
$stmt->bind_param("s", $user_email);
```

Figura 2. Archivo process_modify_user.php (entrega_1)

Aquí, **user_email** proviene de la sesión, lo que vincula la consulta SQL al usuario autenticado.

3.2 INYECCIÓN

En el ámbito de la seguridad informática y el desarrollo de software, la inyección se describe como una vulnerabilidad que da a los atacantes la capacidad de introducir código malicioso en una aplicación o sistema, logrando así su ejecución no autorizada.

- Inyección SQL:

Este tipo de ataque frecuente consiste en que el atacante usa un trozo de código SQL para manipular una base de datos (BD en adelante) y acceder a la información.

En la primera versión del trabajo el código ya incluía algunas consultas parametrizadas correctamente , que es una defensa contra la inyección SQL. Además, una medida que ya se implementó anteriormente en la entrega 1 fue asegurarse de que cada **header("Location: ...")** esté seguido de **exit()**; para evitar que se ejecute cualquier código adicional después de una redirección, previniendo posibles errores o comportamientos inesperados.

De cara a la segunda versión del trabajo se han implementado todas las consultas de manera parametrizada.

Ejemplo de error primera versión del trabajo:

```php
// Consulta para eliminar la canción
$query = "DELETE FROM canciones WHERE id='$id'";
```

Figura 3. archivo process_delete_item.php (entrega_1)

Este código permite al usuario inyectar código SQL malicioso si manipula el valor de id en la URL.

Solución:

```php
// Preparar la consulta SQL para eliminar la canción
$sql = "DELETE FROM canciones WHERE id = ? LIMIT 1";
$stmt = $conn->prepare($sql);
if ($stmt === false) {
        // Registrar el error para revisión interna
        error_log("Error al procesar la solicitud: " . $conn->error);

        // Mostrar mensaje genérico al usuario
        die("Error al procesar la solicitud. Por favor, inténtelo más tarde.");
}

$stmt->bind_param("i", $id);
```

Figura 4. archivo process_delete_item.php (entrega_2)

- Cross-Site Scripting (XSS)

Esta vulnerabilidad ocurre cuando un sitio web permite que alguien inyecte código (como JavaScript) en una página. Si un atacante aprovecha esto, el código malicioso se "refleja" en el navegador de un usuario que visite el enlace afectado, pudiendo ejecutar acciones como robar datos o mostrar contenido no autorizado.

Solución:

Se han escapado los valores que se muestran en HTML usando ***htmlspecialchars()*** y ***strip_tags()*** en los datos del formulario para asegurarnos de que no contengan etiquetas HTML o caracteres especiales.

```php
// Recoger y sanitizar los datos del formulario
$nombre = htmlspecialchars(strip_tags($_POST['nombre']));
$cantante = htmlspecialchars(strip_tags($_POST['cantante']));
$album = htmlspecialchars(strip_tags($_POST['album']));
$genero = htmlspecialchars(strip_tags($_POST['genero']));
$fecha_lanzamiento = htmlspecialchars(strip_tags($_POST['fecha_lanzamiento']));
```

Figura 5. archivo process_add_item.php

- Cross site request forgery (CSRF)

El ataque Cross-Site Request Forgery (CSRF) permite a un atacante, que conoce la URL de un formulario o de una acción específica en el sitio web, enviar solicitudes malintencionadas en nombre de usuarios autenticados. Esto puede llevar a que los usuarios realicen acciones no deseadas sin su consentimiento, como cambios en su perfil, transferencias de datos, o incluso compras no autorizadas.

Solución:

Para prevenir este tipo de ataques, se ha añadido un token CSRF al sitio web. Este token funciona de la siguiente manera:

1. **Generación del token**:
   Cada vez que se carga un formulario o se prepara una acción sensible, el servidor genera un token único y aleatorio vinculado a la sesión del usuario.

```php
// Generar un token CSRF si no existe en la sesión
if (empty($_SESSION['csrf_token'])) {
        $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}

// Obtener el token CSRF para incluirlo en el formulario
$csrf_token = $_SESSION['csrf_token'];
```

Figura 6. archivo process_login.php

2. **Inclusión en las solicitudes**:
   El token se incluye como un campo oculto dentro del formulario.

```php
<h2>Inicio de Sesión</h2>
    <form id="login_form" action="process_login.php" method="POST">
        <input type="hidden" name="csrf_token" value="<?php echo $csrf_token; ?>">
```

Figura 7. archivo process_login.php

3. **Validación en el servidor**:
   Al recibir la solicitud, el servidor verifica que el token incluido en ella coincide con el que se generó inicialmente para esa sesión. Si no coincide, la solicitud se rechaza.

```php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
        // Verificar el token de CSRF
        if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
        die("Error al procesar la solicitud.");
        }
```

Figura 8. archivo process_login.php

3.3 FALLOS CRIPTOGRÁFICOS

En este apartado identificamos fallos en los métodos criptográficos de la web que exponen datos sensibles, tanto en transmisión como en almacenamiento.

- Almacenamiento de las contraseñas

En la primera versión del trabajo no había ningún tipo de mecanismo para encriptar las contraseñas de los usuarios registrados. Para la segunda versión, se almacena un **Hash** a partir de la contraseña facilitada por el usuario y la **Salt** (semilla) del usuario.

Un Hash, es el resultado de aplicar una función de resumen criptográfico a un conjunto de datos. Esta función toma una entrada de tamaño variable y genera una cadena alfanumérica de tamaño fijo, irreversible y único para todas las entradas.

Una Salt**,** es un valor aleatorio que se asigna a cada usuario de la BD.

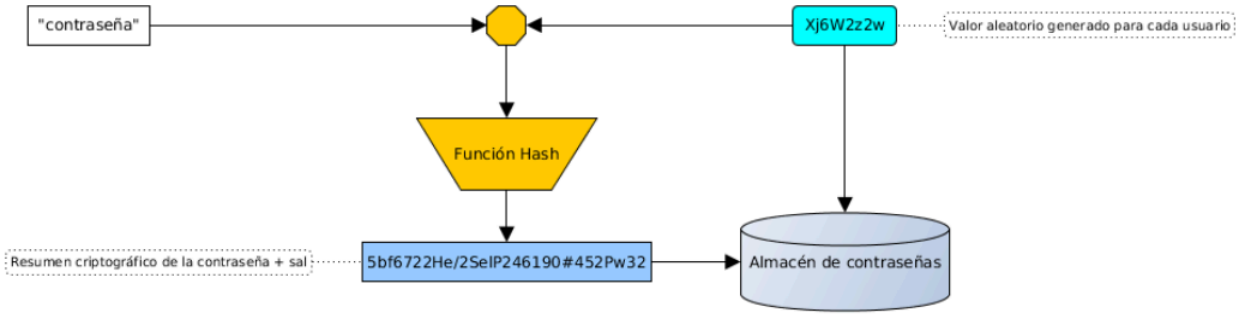

*Figura 9. Creación del **Hash** (contraseña + semilla)*

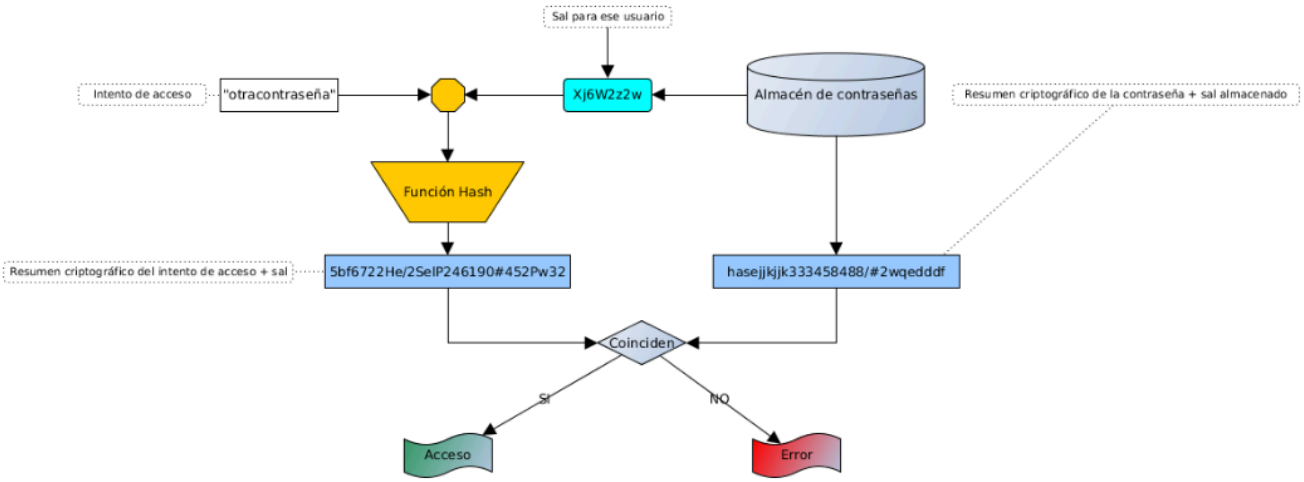

Figura 10. Almacenamiento y verificación

Para ello se ha añadido un nuevo atributo en la tabla de usuarios, llamada semilla para guardar el valor de semilla del usuario. A su vez, se han modificado los archivos php necesarios para gestionar el registro y login.

En *process_register.php* se crea una semilla con el siguiente comando:

bin2hex(random_bytes(16))

Y después de comprobar que las contraseñas introducidas por el usuario son correctas se crea el hash de la semilla con la contraseña con el siguiente comando:

hash('sha256', $contrasenna . $salt);

Una vez obtenido el hash se registra el usuario en la BD junto con su semilla.

En *process_login.php* una vez el usuario introduzca los datos, se busca en la BD al usuario, mediante el email introducido. Si existe, se coje su semilla y se hace un Hash con la contraseña introducida. Para luego compararla con el valor de la BD. Si coinciden se le permite el acceso.

- Almacenamiento de datos sensibles

En caso que nuestra base de datos se haga pública, se ha implementado una encriptación para los datos sensibles.

```php
// Cifrado de datos sensibles
$key = 'clave_secreta_de_32_bytes__';  // Clave secreta para AES-256-CBC
$iv = openssl_random_pseudo_bytes(openssl_cipher_iv_length('aes-256-cbc'));

// Cifrar los datos
$encryptedTelefono = openssl_encrypt($telefono, 'aes-256-cbc', $key, 0, $iv);
$encryptedDni = openssl_encrypt($dni, 'aes-256-cbc', $key, 0, $iv);

// Almacenar el IV y el dato cifrado (en base64 para guardar en la base de datos)
$ciphertextTelefono = base64_encode($iv . $encryptedTelefono);
$ciphertextDni = base64_encode($iv . $encryptedDni);
```

Figura 11. archivo process_register.php

Para ello se ha hecho uso del sistema de cifrado de datos sensibles AES-256-CBC. A su vez se genera una clave fija (que debe ser segura y protegida) y un IV aleatorio para añadir variabilidad. Por último para el almacenamiento seguro el IV y el texto cifrado se almacenan juntos en formato Base64 para facilitar su recuperación y descifrado.

```php
// Función para descifrar los datos
function decryptData($ciphertext)
{
    $key = 'clave_secreta_de_32_bytes__';
    $data = base64_decode($ciphertext);

    $iv = substr($data, 0, openssl_cipher_iv_length('aes-256-cbc'));
    $encryptedData = substr($data, openssl_cipher_iv_length('aes-256-cbc'));

    return openssl_decrypt($encryptedData, 'aes-256-cbc', $key, 0, $iv);
}

// Descifrar valores sensibles
$decryptedTelefono = decryptData($user['telefono']);
$decryptedDni = decryptData($user['dni']);
```

Figura 12. archivo modify_user.php

Teniendo en cuenta que almacenamos los datos sensibles encriptados, es necesario una encriptación para estos datos. Para ello se ha implementado el código que aparece en la figura 12, siendo la funcionalidad de este la siguiente:

- Se decodifica el texto cifrado Base64 para obtener los datos binarios originales.
- Se extrae el IV y los datos cifrados de los binarios.
- Se descifran los datos utilizando la clave secreta y el IV correspondiente.
- Resultado: Se obtiene el valor original en texto plano.

3.4 DISEÑO INSEGURO

El término "diseño inseguro" se utiliza para describir fallos o deficiencias esenciales en la arquitectura o planificación de una aplicación web. Estas fallas hacen que la aplicación sea vulnerable y puedan ser aprovechadas por atacantes para comprometer su funcionamiento, datos o seguridad general.

- Uso de software o configuraciones obsoletas

El proyecto está pensado para una versión anterior de **Ubuntu**. Al principio del proyecto, daban errores relacionados con la versión y había muchas complicaciones para hacerlo funcional a la última; con la 24.04.1. La recomendación fue permanecer en la 22.04.5, para su correcto funcionamiento. (foto eGela)

Aunque seguirá con soporte hasta 2027, la página web será muy vulnerable a largo plazo.

- No aplicar parches o actualizaciones de seguridad

La correcta práctica de los parches o actualizaciones es tener, o intentar tener, todo a la última. El servidor web utiliza las versiones de **mariadb**, 10.11, y **phpMyAdmin**, 5.2.0. Mirando las últimas versiones disponibles, **mariadb** tiene como última versión 11.6.2 , y **phpMyAdmin**, 5.2.1 .

Como en el anterior punto, son versiones especificadas por el profesor, y también se debe al uso de la versión **Ubuntu** 22.04.5. Aun así, tampoco se ha pensado en un proceso de actualización para el sistema web, por lo que queda todavía más expuesto. Para un proyecto que se lleve a cabo y que sea duradero y seguro, se debe tener siempre todo lo más actualizado posible.

- Exposición innecesaria de información del servidor

Desde el inicio se ha incumplido esta práctica. El proyecto está colgado en un repositorio público en **Github**; cualquiera podría acceder al proyecto al completo. Por suerte, es una acción que impuso el profesor de la asignatura. Está hecho a conciencia y en otra circunstancia se hubiera detectado el error para solucionarlo.

Durante el desarrollo del sistema, se observó un problema relacionado con la seguridad de los archivos estáticos. Cuando un usuario accedía directamente a la URL del archivo “validation.js” (localhost:81/src/validation.js), el navegador mostraba el contenido del archivo en formato texto, exponiendo el código JavaScript del cliente. El problema ocurre porque los servidores web, por defecto, permiten la entrega de archivos estáticos como texto o código fuente.

Esto no constituye un problema grave en términos de seguridad, ya que el código JavaScript del lado del cliente siempre está accesible para los usuarios al cargar la página, sin embargo, al permitir la exposición directa del código, se pueden revelar implementaciones detalladas que podrían ser utilizadas para encontrar vulnerabilidades o realizar ingeniería inversa.

Solución:

Se configuró el servidor para permitir el acceso a archivos JavaScript únicamente desde rutas específicas que carguen las páginas autorizadas, impidiendo el acceso directo a través de URL. Esto se ha conseguido incluyendo una regla en el archivo .htaccess que se ha creado para restringir el acceso.

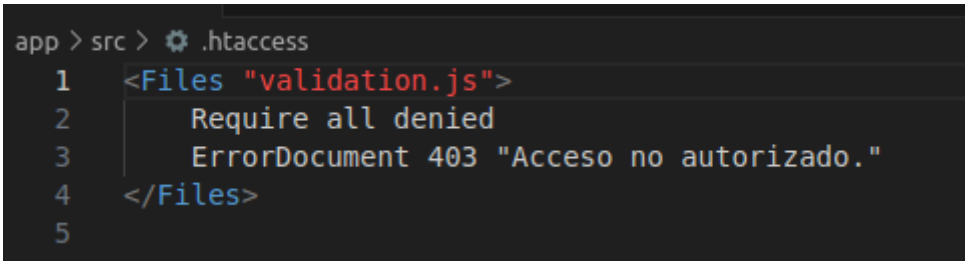

Figura 13. archivo .htaccess

Además se modificó el archivo apache2.conf y en **AllowOverride** en el directorio se cambió de **“All”** a **“None”**. Para conseguir que este cambio se implemente en todos los ordenadores automáticamente sin necesidad de cambiarlo manualmente se introdujo una nueva línea en el archivo Dockerfile del proyecto:

COPY web/conf/apache2.conf /etc/apache2/apache2.conf

Una vez hecho esto, los usuarios ya no pueden acceder directamente al archivo validation.js a través de la URL, protegiendo el código fuente de su exposición innecesaria.

3.5 FALLOS DE IDENTIFICACIÓN Y AUTENTIFICACIÓN

Los fallos de identificación y autenticación son problemas en los procesos destinados a verificar la identidad de un usuario, asegurando que la persona o entidad que intenta acceder a un sistema o recurso sea quien dice ser.

- Captcha

Un CAPTCHA, es un tipo de medida de seguridad conocida como autenticación pregunta-respuesta. Su objetivo es proteger contra el spam y el descifrado de contraseñas mediante una prueba que demuestre que quien responde es un humano y no un ordenador. Además, esta herramienta es ampliamente utilizada para prevenir ataques automatizados, como intentos masivos de inicio de sesión o el abuso de formularios en línea, asegurando la integridad de los servicios web.

Solución:

Hemos añadido a nuestro servidor un captcha, para cuando el usuario quiere registrarse o loguearse, confirme que no es un robot. En este caso hemos implementado la casilla de verificación en el inicio de sesión y en el registro .

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Inicio de Sesión</title>
    <link rel="stylesheet" href="../css/styles.css">
    <script src="https://www.google.com/recaptcha/api.js" async defer></script> <!-- Script de reCAPTCHA -->
</head>
```

Figura 14. añadir captcha al archivo login.php

```php
// Verificar reCAPTCHA
$captcha_secret = '6LcPen4qAAAAAJN8fXyfrXjrgBj7KAX7F5KXw0gN'; // Reemplaza 'YOUR_SECRET_KEY' con tu clave secreta de reCAPTCHA
$captcha_response = $_POST['g-recaptcha-response'];

// Validar la respuesta de reCAPTCHA con la API de Google
$response = file_get_contents("https://www.google.com/recaptcha/api/siteverify?secret=$captcha_secret&response=$captcha_response");
$response_keys = json_decode($response, true);

if (!$response_keys["success"]) {
    $_SESSION['error_message'] = "Error al verificar CAPTCHA. Por favor, intente de nuevo.";
    header("Location: login.php");
    exit();
}
```

Figura 15. añadir captcha al archivo process_login.php y verificación

- Contraseñas débiles o por defecto

Las contraseñas débiles o por defecto son aquellas que resultan fáciles de adivinar o que se emplean de forma repetida en múltiples sistemas. Esto puede facilitar que un atacante logre acceder a un sistema o aplicación de manera no autorizada.

Solución:

Para solucionar este problema, hemos implementado una verificación de contraseñas durante el registro de usuarios. Si la contraseña es demasiado débil, se notificará al usuario y se le solicitará que ingrese una nueva.

```javascript
// Validación de contraseña
const contrasenna = document.getElementById('contrasenna').value;
const contrasennaError = document.getElementById('contrasenna_error');
const contrasennaValida = /^(?=.*[A-Z])(?=.*\d)(?=.*[a-zA-Z])(?=.*[\W_]).{8,}$/;
```

Figura 16. Comprobación de contraseña débil en validation.js

- Gestión de cookies

Si no se configuran adecuadamente los parámetros de cookies en un sistema web, el sistema puede ser vulnerable a una serie de ataques así como robo de Sesión, ataques XSS o ataques MITM(Man-in-the-Middle) entre otros.

Solución:

Se han configurado las cookies de forma que están diseñadas para proteger la sesión del usuario:

- **cookie_secure** protege el identificador de sesión contra ataques de intercepción (por ejemplo, en redes no seguras).
- **cookie_httponly** evita que scripts maliciosos roben las cookies mediante XSS.
- **cookie_lifetime** establece límites claros para que las sesiones no se mantengan indefinidamente activas, reduciendo el riesgo de robo de sesión prolongada.

```php
<?php
session_start([
        'cookie_lifetime' => 86400,
        'cookie_httponly' => true,
        'cookie_secure' => true,
]);
```

Figura 17. archivo process_modify_item.php

3.6 CONFIGURACIÓN DE SEGURIDAD INSUFICIENTE

Una configuración de seguridad incorrecta o insuficiente se refiere a la falta de ajustes adecuados para proteger un sistema, aplicación, red o servicio. Existen diversas áreas en las que la configuración de seguridad puede no estar correctamente implementada, lo que podría hacer que el sistema quede expuesto a riesgos y ataques.

- Gestión de errores

La gestión de los errores en un sistema web pueden revelar configuraciones internas, roles, o esquemas de la BD.

```php
if ($stmt === false) {
    die("Error en la preparación de la consulta: " . $conn->error);
}
```

Figura 18. archivo process_modify_user.php (entrega_1)

El contenido de **$conn->error** puede incluir detalles técnicos sobre la BD o el entorno de ejecución

Un atacante podría usar estos datos para entender la estructura de la BD, ajustar ataques o aprovechar errores de configuración.

Solución:

```php
if ($stmt === false) {
    // Registrar el error para revisión interna
    error_log("Error al procesar la solicitud: " . $conn->error);

    // Mostrar mensaje genérico al usuario
    die("Error al procesar la solicitud. Por favor, inténtelo más tarde.");
}
```

Figura 19. archivo process_modify_item.php

En lugar de mostrar detalles del error al usuario, es mejor registrar el error en un archivo de log seguro para diagnóstico interno y por otra parte, mostrar al usuario un mensaje genérico que no revele detalles sensibles.

- Despliegue seguro

En el archivo *docker-compose.yml* original, las credenciales de la BD estaban directamente visibles. Esta implementación provoca que cualquiera con acceso al repositorio o al archivo **.***docker-compose.yml* pueda ver las credenciales de la BD. Además, si el archivo se sube a un repositorio público, estas credenciales estarán expuestas a todo el mundo.

Solución:

En la segunda entrega, las credenciales de la BD se movieron a un archivo *.env***:**

```
MYSQL_USER: admin
MYSQL_PASSWORD: sgssi_proyecto
MYSQL_DATABASE: database
```

Figura 20. archivo .env (entrega_2)

```yaml
phpmyadmin:
  image: phpmyadmin/phpmyadmin:5.2.0 # Cambia a una versión especifica en lugar de latest
  links:
    - db
  ports:
    - 8890:80
  environment:
    MYSQL_USER: ${MYSQL_USER}
    MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    MYSQL_DATABASE: ${MYSQL_DATABASE}
```

Figura 21. archivo docker-compose.yml (entrega_2)

El archivo **.env** debería añadirse a **.gitignore** para evitar que se suba al repositorio, pero en este caso se ha subido para que sea visible en la entrega.

- Cabecera CSP

Las cabeceras CSP (Content Security Policy) son una herramienta de seguridad en la programación web diseñada para reducir los riesgos de ataques como el Cross-Site Scripting (XSS) y otras formas de inyección de código malicioso en sitios web.

Solución:

Para arreglar el problema, hemos añadido una cabecera CSP con los siguientes valores:

```
<html>
<head>
<meta http-equiv="Content-Security-Policy"
      content="default-src 'none';
               script-src 'self' 'unsafe-inline';
               style-src 'self' 'unsafe-inline';
               img-src 'self' ;
               form-action 'self';">

</head>
```

Figura 22. archivo Cabecera CSP (entrega_2)

Con **'unsafe-inline'** es menos seguro porque puede permitir inyecciones de código

pero si no, no se cargan el reCAPTCHA.

- Cabecera X-Content-Type-Options

El encabezado HTTP X-Content-Type-Options es una instrucción enviada por el servidor que le indica al navegador que debe confiar y respetar el tipo MIME declarado en el encabezado Content-Type, sin intentar adivinarlo o cambiarlo. Esto ayuda a prevenir problemas de seguridad relacionados con el *MIME-sniffing*, asegurando que el contenido sea interpretado únicamente de la manera prevista por el servidor.

Solución:

Para arreglar el problema, hemos añadido una cabecera 'X-Content-Type-Options' con los siguientes valores:

```
Header always set X-Content-Type-Options "nosniff"
```

Figura 23. archivo .htaccess (entrega_2)

- Cabecera X-Frame-Options

Si un sitio web malicioso puede insertar tu página dentro de un iframe, podría utilizar esta técnica para engañar al usuario y hacer que realice acciones no deseadas mediante un ataque conocido como *clickjacking*. Además, ciertos ataques, como *Spectre*, podrían permitir a estos sitios maliciosos acceder al contenido de un documento incrustado.

La cabecera X-Frame-Options le indica al navegador si una página puede ser mostrada dentro de un <frame>, <iframe>, <embed> o <object>.

Solución:

Para arreglar el problema, hemos añadido una cabecera 'X-Frame-Options' con los siguientes valores:

```
Header set X-Frame-Options "DENY"
```

Figura 24. archivo .htaccess (entrega_2)

- Cabecera HSTS

La comunicación mediante una conexión HTTP sin cifrar no está protegida, lo que significa que los datos enviados pueden ser interceptados y leídos por terceros en la red.

La cabecera Strict-Transport-Security (HSTS) le indica al navegador que siempre debe acceder al sitio utilizando HTTPS, evitando que se cargue mediante HTTP. Una vez configurada esta cabecera, el navegador forzará el uso de HTTPS para acceder al dominio, sin necesidad de redirigirlo, durante el período de tiempo especificado en la cabecera.

Solución:

Para arreglar el problema, hemos añadido una cabecera HSTS

con los siguientes valores:

```
Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains;
```

Figura 25. archivo .htaccess (entrega_2)

- Cabecera X-XSS-Protection

La cabecera X-XSS-Protection impide la carga de una página cuando detecta ataques del tipo Cross-Site (XSS). Esta protección ya no es necesaria en los navegadores modernos cuando el sitio implementa una fuerte Content-Security-Policy que deshabilita el uso de Javascript inline ('unsafe-inline'). Sin embargo da protección a los usuarios de navegadores más antiguos que no soportan CSP.se utilizan para controlar y mitigar estos ataques.

Solución:

Para arreglar el problema, hemos añadido una cabecera 'X-XSS-Protection' con los siguientes valores:

```
Header always set X-XSS-Protection "1; mode=block"
```

Figura 26. archivo .htaccess (entrega_2)

- Cabecera Referrer-Policy

El encabezado de respuesta HTTP controla la cantidad de información de referencia que se incluye en las solicitudes cuando el navegador realiza una petición a otro sitio. Además de esta configuración a nivel de servidor, también puedes establecer esta política directamente en el código HTML de la página.

Solución:

Para arreglar el problema, hemos añadido una cabecera 'Referrer-Policy'' con los siguientes valores:

```
Header always set Referrer-Policy "no-referrer"
```

Figura 27. archivo .htaccess (entrega_2)

- Cache-Control

La cabecera **`Cache-Control`** controla cómo y por cuánto tiempo los recursos web (como páginas HTML, imágenes, scripts, etc.) Se almacenan en caché tanto en los navegadores de los usuarios como en los servidores intermedios (como proxies y CDN). Su principal objetivo es mejorar la velocidad de carga de las páginas web y reducir la carga en los servidores, pero también puede ayudar a gestionar la validez de los recursos y la seguridad de los mismos.

Solución:

Para solucionar este problema, crearemos una cabecera 'Cache-Control' con el valor 'no-store' para que el navegador no almacene en caché la página.

```
<meta http-equiv="Cache-Control" content="no-store, no-cache, must-revalidate
```

Figura 28. archivo registrer.php (entrega_2)

- Expect-CT

Ayuda a proteger contra ataques de *Certificate Transparency* al permitir que los navegadores detecten certificados SSL/TLS no registrados en los logs públicos de transparencia.

Solución:

Para arreglar el problema, hemos añadido una cabecera 'Expect-CT'' con los siguientes valores:

```
Header always set Expect-CT "enforce, max-age=86400"
```

Figura 29. archivo .htaccess (entrega_2)

3.7 FALLOS EN LA INTEGRIDAD DE DATOS Y SOFTWARE

Ocurre cuando el sistema no protege adecuadamente la exactitud, consistencia o confiabilidad de los datos y el software . Puede llevar a pérdida de confianza en el sistema, mal funcionamiento del software, exposición de información sensible, o incluso comprometer la seguridad completa de la aplicación.

- Pipeline CI/CD

Se ha generado un pipeline de CI/CD en GitHub Actions mediante los Workflows. Este pipeline asegura la calidad del software al automatizar pruebas y configuraciones críticas. Previene fallos en la integridad de datos y software al:

- Detectar errores tempranos.
- Garantizar que las dependencias y el entorno sean consistentes y seguros.
- Automatizar pasos críticos para eliminar el riesgo de errores humanos.

```yaml
jobs:
  test:
        runs-on: ubuntu-latest

         steps:
          # 1. Checkout del código
          - name: Checkout code
            uses: actions/checkout@v4




          # 2. Configuración de PHP
            - name: Set up PHP
              run: |
                sudo apt-get update
                sudo apt-get install -y php php-cli php-xml php-mbstring




            # 3. Instalar Composer
            - name: Install Composer
              run: |
                curl -sS https://getcomposer.org/installer | php
                sudo mv composer.phar /usr/local/bin/composer



            # 4. Instalar dependencias de Composer
            - name: Install dependencies (Composer)
              run: |
                composer install



            # 5. Descargar PHPUnit (si no está instalado globalmente)
            - name: Install PHPUnit
              run: |
                curl -sS https://phar.phpunit.de/phpunit.phar -o phpunit.phar
                chmod +x phpunit.phar
                sudo mv phpunit.phar /usr/local/bin/phpunit

            # 6. Ejecutar las pruebas con PHPUnit
            - name: Run PHP tests (usando PHPUnit)
              run: |
                php phpunit.phar --configuration phpunit.xml  # O simplemente usa "phpunit" si ya     está en el $PATH
```

Figura 30. archivo ci-cd-pipeline.yml

3.8 FALLOS EN LA MONITORIZACIÓN DE LA SEGURIDAD

Esta vulnerabilidad se refiere a la falta de supervisión constante y adecuada de los sistemas, aplicaciones o redes que podrían ayudar a detectar y prevenir posibles amenazas de seguridad. Como resultado, los administradores no tienen visibilidad sobre actividades sospechosas o incidentes de seguridad, lo que puede hacer que los problemas pasen desapercibidos hasta que ya haya un daño significativo.

Solución:

La forma de prevenir este problema es creando copias de seguridad de la base de datos regularmente. En este caso, se puede acceder a la base de datos usando phpMyAdmin y exportar los datos en un archivo .sql para guardarlos y poder recuperarlos si es necesario.

Otro aspecto a tener en cuenta sería desarrollar un plan de seguridad para el sistema y llevar a cabo auditorías periódicas para verificar que no surjan nuevos problemas de seguridad y mantener el sistema protegido.

4. SEGUNDA AUDITORÍA

Una vez arreglados los anteriores fallos de seguridad identificados, se ha vuelto a realizar una segunda auditoría con la misma herramienta que en la primera, ZAP, para comprobar si se han obtenido mejoras en cuanto a la seguridad de la web y no se han detectado más riesgos.

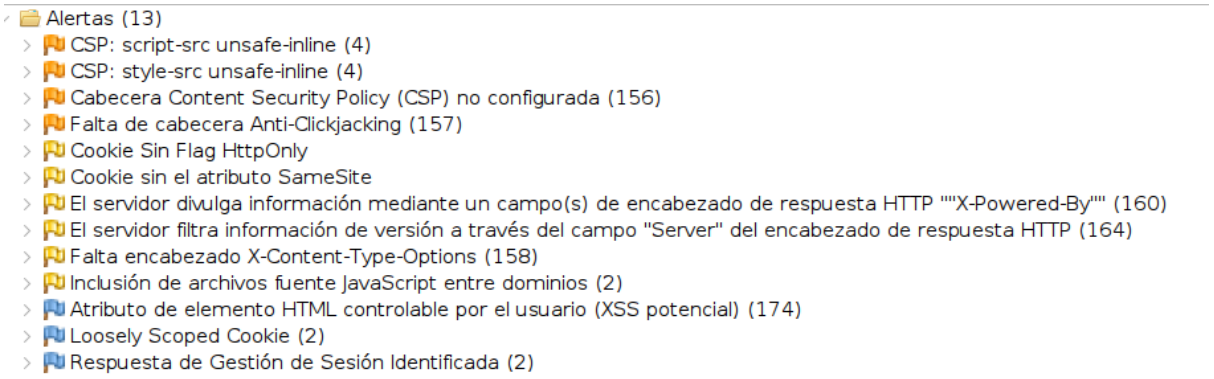

Figura 31. Errores de la segunda auditoría

Al ejecutarla en el sitio web, fueron identificados un total de 13 errores. A pesar de seguir teniendo varios errores, los más importantes han sido solucionados, como puede ser la inyección SQL.

Sin embargo, una gran parte de estos errores son dados porque ZAP no puede comprobar algunas de las librerías que usamos en nuestro proyecto como reCAPTCHA.

5. CONCLUSIONES

A lo largo de este análisis de seguridad del sistema web *MusicCloud*, se han identificado diversas vulnerabilidades críticas y áreas de mejora que comprometen la seguridad de la aplicación. En esta segunda entrega, muchas de ellas han sido corregidas.

El proceso comenzó con una auditoría de seguridad detallada en la que utilizamos herramientas como ZAP para identificar vulnerabilidades comunes, como inyecciones SQL, XSS, CSRF y fallos de autenticación. Los resultados iniciales mostraron que el sistema estaba expuesto a múltiples riesgos debido a prácticas de programación inseguras, falta de medidas criptográficas adecuadas, y configuraciones de seguridad insuficientes.

Una de las acciones más relevantes fue la implementación de consultas SQL parametrizadas y la introducción de una capa de protección adicional mediante tokens CSRF. También se mejoró la seguridad de almacenamiento de contraseñas utilizando un hash con *salt*, asegurando que las contraseñas ya no se almacenan en texto plano.

En cuanto al diseño y la gestión de la aplicación, se identificaron errores relacionados con la exposición de información sensible, como la falta de cifrado adecuado en las credenciales y la exposición de detalles técnicos en los mensajes de error. Estos problemas fueron resueltos mediante el uso de archivos de configuración más seguros y la mejora en la gestión de errores y sesiones.

El uso de un pipeline CI/CD también ha permitido automatizar procesos cruciales para la seguridad del código, como la ejecución de pruebas y la validación de dependencias, reduciendo el riesgo de fallos en la integridad del software.

A pesar de las mejoras significativas realizadas, aún existen áreas en las que el sistema puede seguir evolucionando. Es fundamental mantener actualizados los componentes del sistema, realizar auditorías periódicas y fortalecer las prácticas de seguridad en todas las fases del desarrollo.

En resumen, este proyecto demuestra que es posible transformar una aplicación web vulnerable en un sistema más seguro mediante la aplicación de buenas prácticas de seguridad informática. Las lecciones aprendidas de este análisis permitirán que el sistema *MusicCloud* se encuentre más protegido ante posibles ataques.

6. BIBLIOGRAFÍA

[1] Mozilla Developer Network
Documentación sobre encabezados HTTP
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers (accedido el 11/2024)

[2] web.dev
Guía sobre cómo mejorar la seguridad con encabezados HTTP
https://web.dev/articles/security-headers?hl=es (accedido el 11/2024)

[3] Kinsta
Cómo configurar un pipeline CI/CD
https://kinsta.com/es/blog/como-configurar-pipeline-ci-cd/ (accedido el 11/2024)

[4] ChatGPT
Generador de texto basado en IA
https://chatgpt.com/ (accedido el 11/2024)

[5] DesarrolloWeb
Cómo poner un CAPTCHA en 3 pasos
https://desarrolloweb.com/articulos/poner-captcha-en-3-pasos.html (accedido el 11/2024)

[6] OWASP
Proyecto OWASP Top Ten
https://owasp.org/www-project-top-ten/ (accedido el 11/2024)