

eman ta zabal zazu



Universidad  
del País Vasco

Euskal Herriko  
Unibertsitatea

# Sistemas de Gestión de Seguridad de Sistemas de Información

Ingeniería Informática de Gestión y Sistemas de  
Información

## Sistema Web

Autores:

Xabier Gabiña  
Ainhize Martinez  
Marcos Martín

5 de noviembre de 2023

# Índice general

<b>1. Introduccion</b>	<b>3</b>
<b>2. Primera auditoria</b>	<b>4</b>
2.1. ZAP . . . . .	4
2.2. sqlmap . . . . .	4
2.3. Metaexploit . . . . .	4
<b>3. Vulnerabilidades</b>	<b>5</b>
3.1. Rotura de control de acceso . . . . .	5
3.1.1. Control de acceso . . . . .	5
3.1.2. Configuracion erronea de las Cookies . . . . .	8
3.2. Fallos criptográficos . . . . .	9
3.2.1. Forzar HTTPS . . . . .	9
3.2.2. Deshabilitar el almacenamiento en cache . . . . .	9
3.2.3. Almacenamiento de contraseñas . . . . .	9
3.3. Inyección . . . . .	11
3.3.1. Procesado de consultar SQL . . . . .	11
3.4. Diseño inseguro . . . . .	12
3.4.1. Auditorias de seguridad . . . . .	12
3.4.2. Reutilizacion de codigos seguros . . . . .	12
3.5. Configuración de seguridad insuficiente . . . . .	13
3.5.1. Entornos de desarrollo . . . . .	13
3.5.2. Plataforma minima . . . . .	13
3.5.3. Despliegue seguro . . . . .	13
3.5.4. Cabeceras CSP . . . . .	13
3.6. Componentes vulnerables y obsoletos . . . . .	14
3.6.1. Control de versiones de los componentes . . . . .	14
3.7. Fallos de identificación y autenticación . . . . .	15
3.7.1. Evitar ataques automatizados . . . . .	15
3.7.2. Contraseñas debiles o por defecto . . . . .	15
3.7.3. Autenticacion de dos factores . . . . .	15
3.7.4. Invalidacion de sesiones . . . . .	15
3.8. Fallos en la integridad de datos y software . . . . .	16
3.8.1. Firmas digitales . . . . .	16

3.8.2.	Bibliotecas y dependencias confiables . . . . .	16
3.8.3.	Uso de herramientas de analisis . . . . .	16
3.9.	Fallos en la monitorizacion de la seguridad . . . . .	17
3.9.1.	Implementacion de un log . . . . .	17
3.10.	Falsificacion de Solicitud del Lado del Servidor (SSRF) . . . . .	18
3.10.1.	Control del trafico . . . . .	18
3.10.2.	Asegurar el codigo . . . . .	18
3.11.	Problemas de calidad de codigo . . . . .	19
3.11.1.	Revision de calidad del codigo . . . . .	19
3.12.	Problemas de denegacion de servicios . . . . .	20
3.12.1.	Pruebas de rendimiento . . . . .	20
<b>4.</b>	<b>Segunda auditoria</b>	<b>21</b>
<b>5.</b>	<b>Conclusiones</b>	<b>22</b>
<b>6.</b>	<b>Bibliografia</b>	<b>23</b>

# Introduccion

# Primera auditoria

La idea de esta auditoria es la de encontrar los fallos de seguridad que tiene nuestro sistema web para en el posterior capitulo de este documento comentarlos y solucionarlos.

## 2.1. ZAP

Para empezar con la primera auditoria ejecutaremos el proxy ZAP como se nos ha propuesto en clase. Al ejecutarla en nuestra pagina web nos encontramos con el siguiente listado de errores:

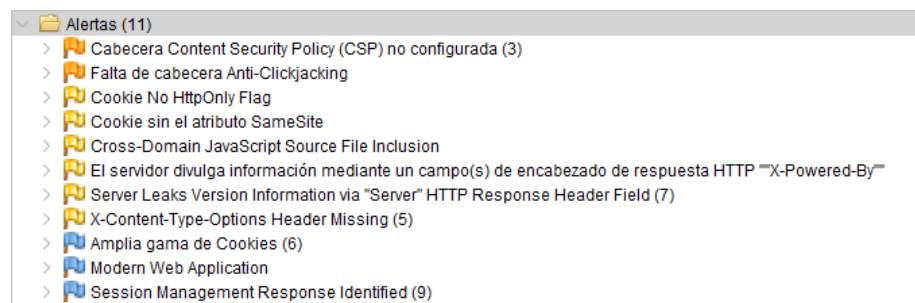


Figura 2.1: Listado de errores de la primera auditoria

Como podemos ver en la imagen, tenemos un total de 11 errores, los cuales iremos comentando uno a uno en los siguientes apartados y solucionando.

## 2.2. sqlmap

## 2.3. Metaexploit

# Vulnerabilidades

## 3.1. Rotura de control de acceso

La rotura de control de acceso es una vulnerabilidad que permite a un atacante acceder a recursos restringidos o privilegiados, ya sea por un error en la implementación de la autenticación y autorización o por un error en la lógica de control de acceso. Dentro de nuestro sistema tenemos varios fallos de rotura de control y ahora hablaremos de ellas y de como las hemos solucionado.

### 3.1.1. Control de acceso

#### Descripción

En nuestro sistema, un usuario puede modificar sus datos personales, pero también puede modificar los datos de otros usuarios. Esto es un fallo de rotura de control de acceso ya que un usuario no debería poder modificar los datos de otro usuario.

#### PoC

Pongamos en el ejemplo que tenemos dos usuarios, Admin y Xabier con sus respectivas IDs

id	nombre
1	Xabier
10	Admin

Figura 3.1: Datos de Usuarios

Si desde inspeccionar elementos hacemos click sobre el boton 'Perfil' este nos mostrara el link el cual se ve asi:

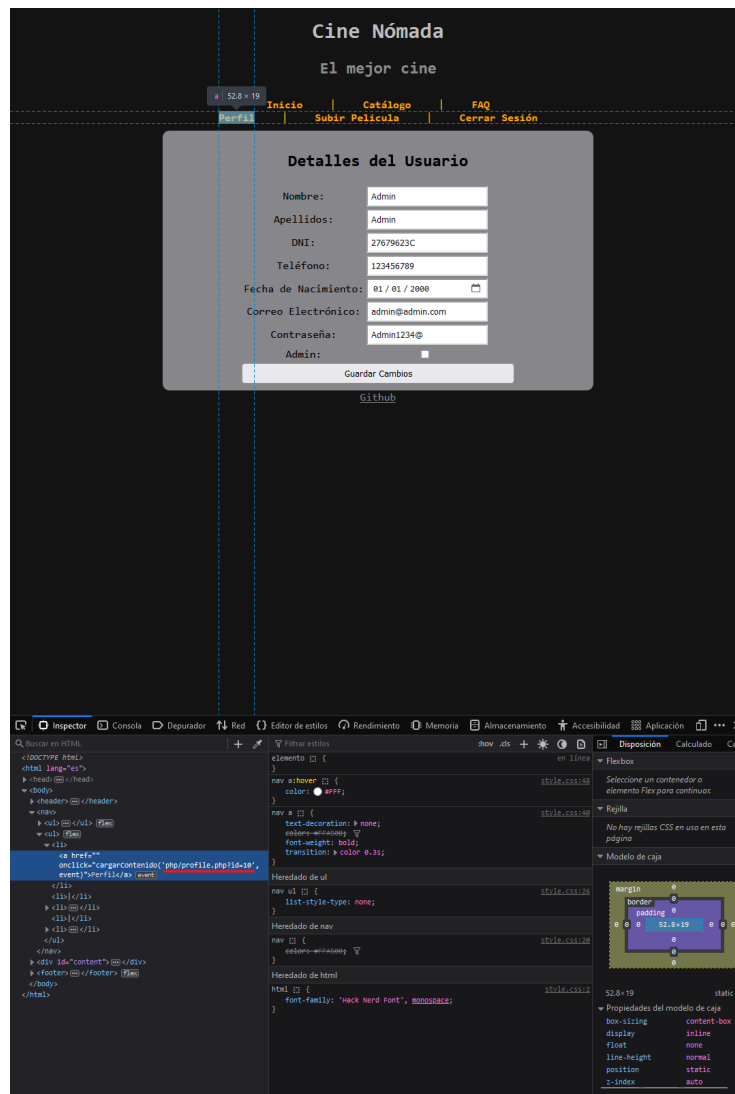


Figura 3.2: Perfil de Admin

Si alteramos el valor de `?id=X` por en este caso la id de Xabier (La ID 1) podemos acceder a sus datos

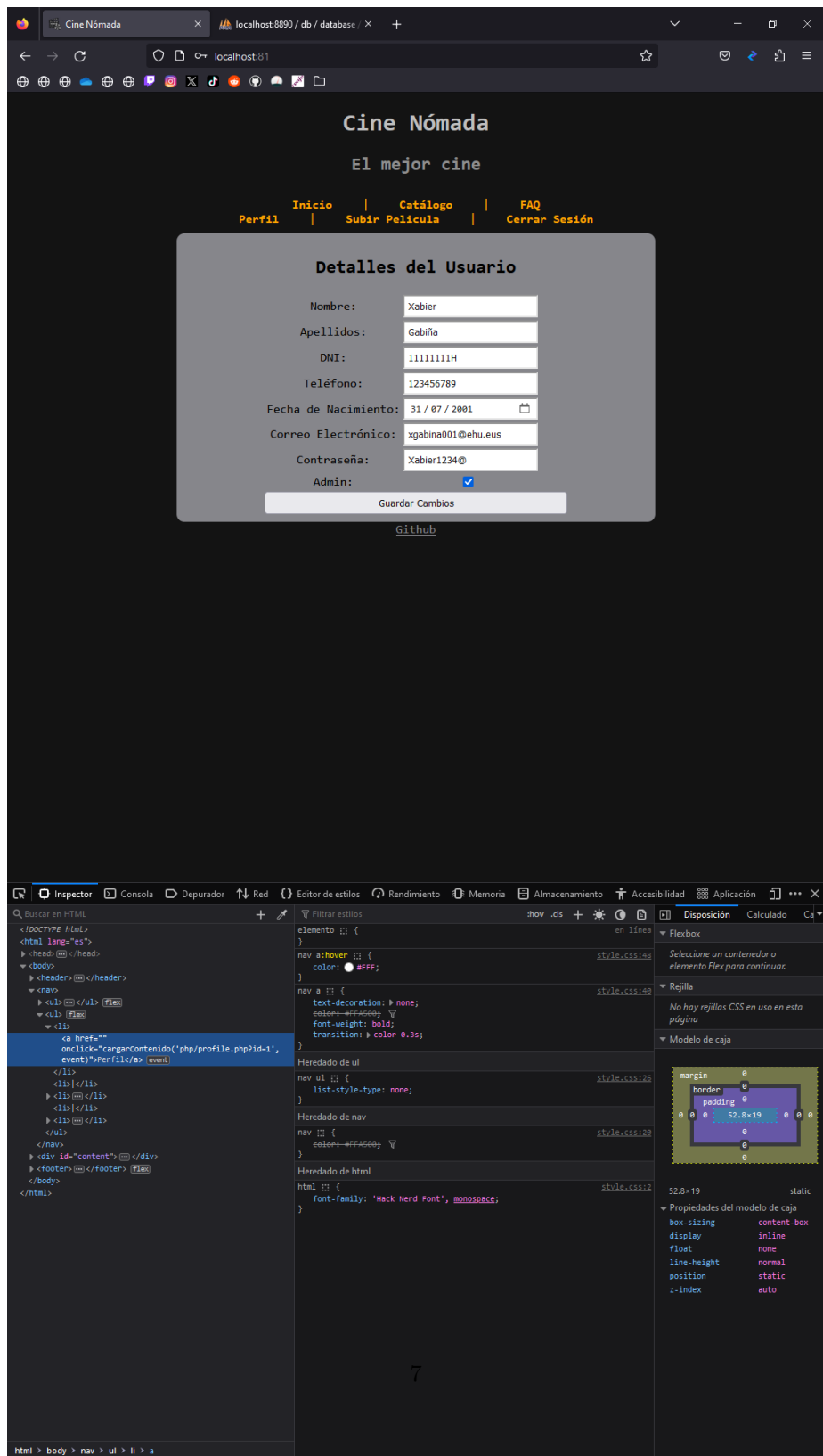


Figura 3.3: Perfil de Xabier



## Solución

Para solucionar este problema, hemos añadido una comprobación en el código que comprueba que el usuario que está intentando modificar los datos es el mismo que el usuario que está logueado en el sistema.

```
// Verificar si se recibió un ID válido a través de la URL
if (isset($_GET['id']) && is_numeric($_GET['id']))
{
    // Verificar si el usuario es el mismo que el de la sesión
    if ((int)$_GET['id'] == (int)$_SESSION['user_id'])
    {
        $userId = $_SESSION['user_id'];
    }
}
```

Figura 3.4: Comprobación de usuario

Esta misma error tambien ha sido corregido en el catalogo.

### 3.1.2. Configuración errónea de las Cookies

#### Descripción

En nuestro sistema, las cookies no tienen la configuración segura, lo que permite que un atacante pueda obtener información sensible de los usuarios.

#### Solución

Para solucionar este problema, hemos añadido la configuración segura a las cookies.

```
<?php
session_start([
    'cookie_lifetime' => 0,           // La sesión expira cuando se cierra el navegador.
    'cookie_path' => '/',             // Disponible en todo el dominio.
    'cookie_secure' => true,          // Solo se envía la cookie sobre conexiones HTTPS.
    'cookie_httponly' => true,        // La cookie solo es accesible a través de HTTP.
    'cookie_samesite' => 'Lax',       // Define la política de SameSite (puede ser 'Lax' o 'Strict').
]);
?>
```

Figura 3.5: Configuración de las cookies

## 3.2. Fallos criptográficos

### 3.2.1. Forzar HTTPS

#### Descripción

En nuestro sistema, no se fuerza el uso de HTTPS, lo que permite que un atacante pueda interceptar el tráfico de la página y obtener información sensible de los usuarios.

#### Solución

Para solucionar este problema configuraremos nuestro servidor para que cifre y redirija todo el tráfico a HTTPS.

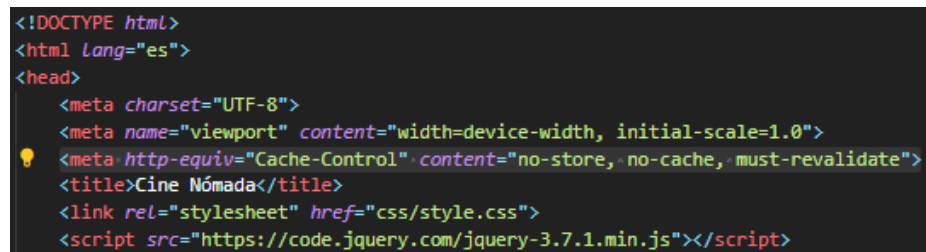
### 3.2.2. Deshabilitar el almacenamiento en cache

#### Descripción

En nuestro sistema, no se deshabilita el almacenamiento en cache, lo que permite que un atacante pueda obtener información sensible de los usuarios.

#### Solución

Para solucionar este problema crearemos una cabecera `Cache-Control` con el valor `"no-store"` para que el navegador no almacene en cache la página.

A screenshot of a code editor showing HTML code. The code includes a DOCTYPE declaration, an html tag with lang="es", and a head section. Inside the head, there are meta tags for charset="UTF-8" and viewport. A meta tag with http-equiv="Cache-Control" and content="no-store, no-cache, must-revalidate" is highlighted with a yellow background. Below it are title, link, and script tags.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="Cache-Control" content="no-store, no-cache, must-revalidate">
  <title>Cine Nómada</title>
  <link rel="stylesheet" href="css/style.css">
  <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
```

Figura 3.6: Cabecera Cache-Control

### 3.2.3. Almacenamiento de contraseñas

#### Descripción

En nuestro sistema, no se almacenan las contraseñas de los usuarios de forma segura, lo que permite que un atacante pueda obtener las contraseñas de los usuarios.

### PoC

Si un atacante consigue acceso a la base de datos, puede obtener las contraseñas de los usuarios en texto plano.

email	passwd
xabierland@gmail.com	Xabier1234@
admin@admin.com	Admin1234@

Figura 3.7: Contraseñas en texto plano

### Solución

Para solucionar este problema, hemos añadido una función que cifra las contraseñas de los usuarios antes de almacenarlas en la base de datos.

```
// Hashear la contraseña
$options=['cost'=>12,];
$hashedPassword = password_hash($passwd, PASSWORD_BCRYPT, $options);
```

Figura 3.8: Contraseñas cifradas

## 3.3. Inyección

### 3.3.1. Procesado de consultar SQL

#### Descripción

En nuestro sistema, no se procesan correctamente las consultas SQL, lo que permite que un atacante pueda obtener información sensible de los usuarios.

#### PoC

#### Solución

Para solucionar este problema, hemos modificado el código que procesa las consultas SQL para que no se puedan inyectar consultas SQL.

```
// SQL seguro utilizando consultas preparadas
$sql = "INSERT INTO usuarios (nombre, apellidos, passwd, dni, fechaN, email, telefono) VALUES (?, ?, ?, ?, ?, ?, ?)";

if ($stmt = $conn->prepare($sql)) {
    // Vincula las variables a los marcadores de posición
    $stmt->bind_param("sssssss", $nombre, $apellidos, $hashedPassword, $dni, $fechaNacimiento, $email, $telefono);

    // Ejecuta la consulta preparada
    if ($stmt->execute()) {
        echo "Registrado con éxito";
    } else {
        echo "Error al ejecutar la consulta: " . $stmt->error;
    }
} else {
    echo "Error al preparar la consulta: " . $conn->error;
}
```

Figura 3.9: Parametrizar consulta SQL

### **3.4. Diseño inseguro**

#### **3.4.1. Auditorias de seguridad**

#### **3.4.2. Reutilizacion de codigos seguros**

## **3.5. Configuración de seguridad insuficiente**

### **3.5.1. Entornos de desarrollo**

### **3.5.2. Plataforma minima**

### **3.5.3. Despliegue seguro**

### **3.5.4. Cabeceras CSP**

#### **Descripción**

Establecer una CSP en un sitio web sirve para mejorar la seguridad de tu aplicación web y reducir el riesgo de ataques de seguridad. Una CSP especifica las fuentes de las que se pueden cargar recursos (como scripts, estilos, imágenes, etc.) en una página web.

#### **Solución**

Para solucionar este problema, hemos añadido una cabecera Content-Security-Policy con los siguientes valores:

También hemos añadido X-Frame-Options para evitar ataques de clickjacking.

## **3.6. Componentes vulnerables y obsoletos**

### **3.6.1. Control de versiones de los componentes**

#### **Descripción**

En nuestro sistema no se utilizan las ultimas versiones de todos los componentes lo que puede resultar en una brecha de seguridad.

#### **Solución**

Para solucionar este problema, hemos actualizado todos los componentes a sus ultimas versiones.

- PHP 7.2.2 → 8.2
- MariaDB 10.8.2 → 11.1.2
- phpMyAdmin ya estaba en la ultima version.
- jQuery 3.6.0 → 3.7.1

### **3.7. Fallos de identificación y autenticación**

**3.7.1. Evitar ataques automatizados**

**3.7.2. Contraseñas debiles o por defecto**

**3.7.3. Autenticacion de dos factores**

**3.7.4. Invalidacion de sesiones**



### **3.8. Fallos en la integridad de datos y software**

#### **3.8.1. Firmas digitales**

#### **3.8.2. Bibliotecas y dependencias confiables**

#### **3.8.3. Uso de herramientas de analisis**

### **3.9. Fallos en la monitorizacion de la seguridad**

#### **3.9.1. Implementacion de un log**

### **3.10. Falsificacion de Solicitud del Lado del Servidor (SSRF)**

**3.10.1. Control del trafico**

**3.10.2. Asegurar el codigo**

### **3.11. Problemas de calidad de código**

#### **3.11.1. Revisión de calidad del código**

### **3.12. Problemas de denegacion de servicios**

#### **3.12.1. Pruebas de rendimiento**

## Segunda auditoria

# Conclusiones

# Bibliografia

- OWASP. (2021). Informe de Vulnerabilidades. OWASP. <https://owasp.org/www-project-top-ten/>
- GPT-3.5. (2023). Respuestas a preguntas varias. OpenAI. <https://www.openai.com/>
- GitHub Copilot. (2022). Autocompletado. GitHub. <https://github.com/features/copilot>