

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

7 de noviembre de 2023

Índice general

1. Introduccion	3
2. Primera auditoria	4
2.1. ZAP	4
2.2. sqlmap	5
2.3. nmap	5
2.4. Metaexploit	5
3. Vulnerabilidades	6
3.1. Rotura de control de acceso	6
3.1.1. Control de acceso	6
3.2. Fallos criptográficos	10
3.2.1. Cifrado de extremo a extremo	10
3.2.2. Almacenar informacion sensible	11
3.2.3. Almacenamiento de contraseñas	12
3.2.4. Configuracion erronea de las Cookies	14
3.3. Inyección	15
3.3.1. Procesado de consultar SQL	15
3.4. Diseño inseguro	16
3.4.1. Auditorias de seguridad	16
3.4.2. Reutilizacion de codigos seguros	17
3.5. Configuración de seguridad insuficiente	18
3.5.1. Entornos de desarrollo	18
3.5.2. Plataforma minima	19
3.5.3. Despliegue seguro	20
3.5.4. Cabeceras CSP	21
3.5.5. Cabeceras HSTS	22
3.5.6. Cabeceras X-XSS-Protection	23
3.5.7. Cabeceras X-Content-Type-Options	24
3.5.8. Cabeceras X-Frame-Options	25
3.5.9. Cabeceras Cache-Control	26
3.5.10. Configuracion PHP	27
3.5.11. Configuracion Apache	28
3.6. Componentes vulnerables y obsoletos	29

3.6.1.	Control de versiones de los componentes	29
3.7.	Fallos de identificación y autenticación	30
3.7.1.	Autenticacion de dos factores	30
3.7.2.	Contraseñas debiles o por defecto	31
3.7.3.	Invalidacion de sesiones	32
3.8.	Fallos en la integridad de datos y software	33
3.8.1.	Firmas digitales	33
3.8.2.	Bibliotecas y dependencias confiables	34
3.8.3.	Uso de herramientas de analisis	35
3.9.	Fallos en la monitorizacion de la seguridad	36
3.9.1.	Configuracion de logs del sistema	36
3.9.2.	Implementacion de un log	37
3.10.	Falsificacion de Solicitud del Lado del Servidor (SSRF)	38
3.10.1.	Control del trafico	38
3.10.2.	Validacion de accesos	39
3.10.3.	Asegurar el codigo	40
3.11.	Problemas de calidad de codigo	41
3.11.1.	Revision de calidad del codigo	41
3.12.	Problemas de denegacion de servicios	42
3.12.1.	Pruebas de rendimiento	42
4.	Segunda auditoria	43
4.1.	ZAP	43
4.2.	sqlmap	43
4.3.	nmap	43
4.4.	Metaexploit	43
5.	Conclusiones	44
6.	Bibliografia	45

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:

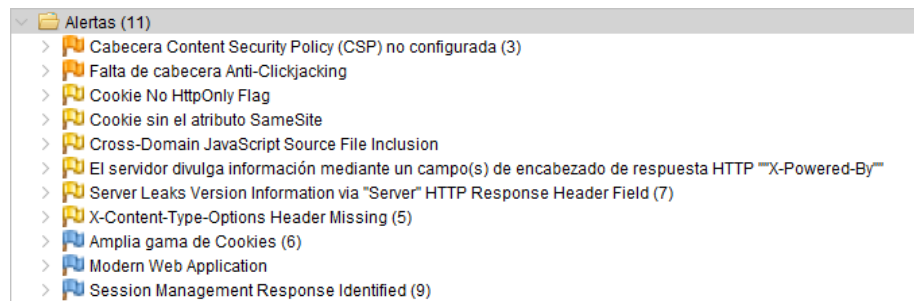


Figura 2.1: Listado de errores de la primera auditoria

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. nmap

2.4. Metasploit

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:

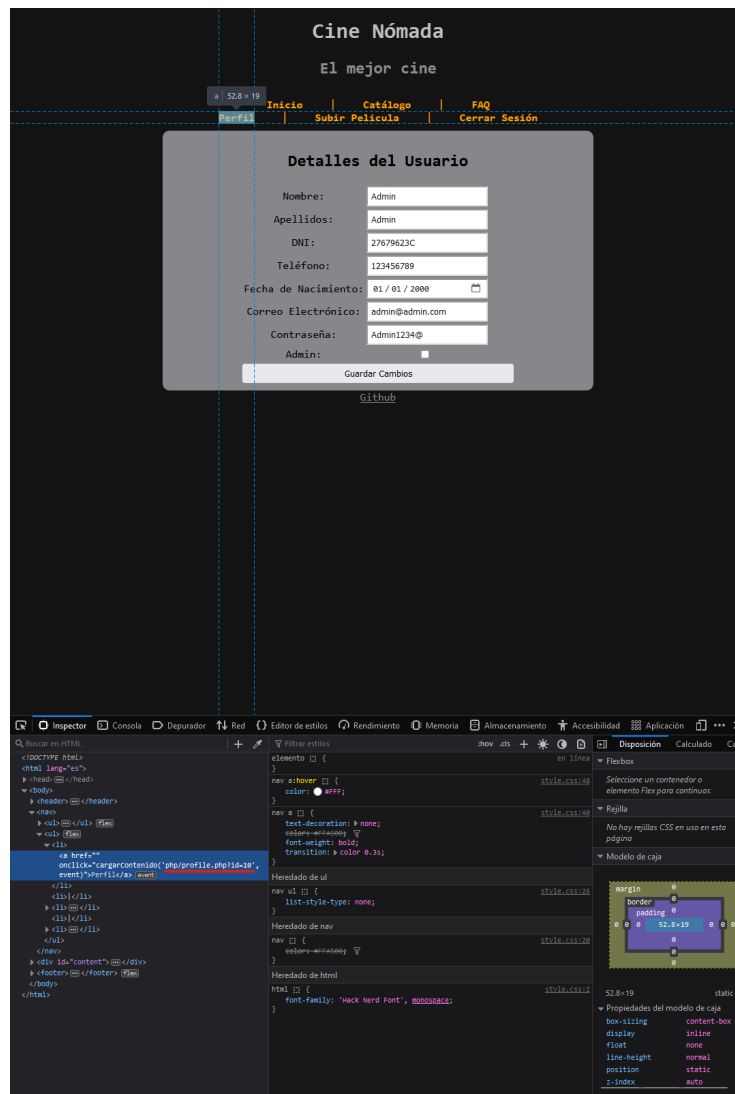


Figura 3.2: Perfil de Admin

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

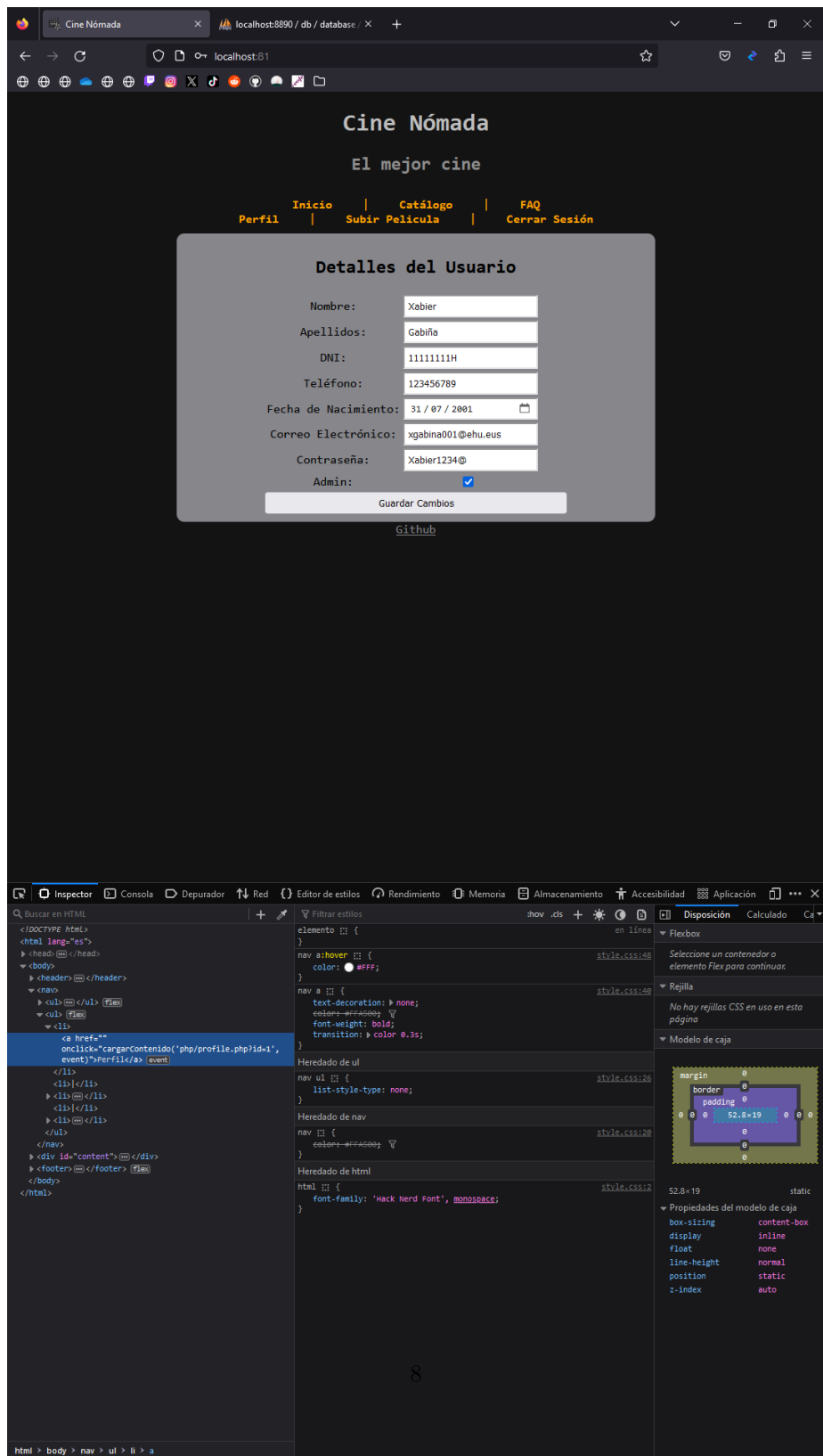


Figura 3.3: Perfil de Xabier

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.

3.2. Fallos criptográficos

3.2.1. Cifrado de extremo a extremo

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.

Creamos un certificado SSL autofirmado dentro del Dockerfile.

Creamos una regla para redirigir el tráfico entrante del puerto 80 al puerto 443.

```
<VirtualHost *:80>
    ServerName localhost
    Redirect / https://localhost/
</VirtualHost>
```

Figura 3.5: Redirección de tráfico

También debemos decir al puerto 443 que use el certificado SSL que hemos creado.

```
<VirtualHost *:443>
    ServerName localhost
    DocumentRoot /var/www/html

    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
</VirtualHost>
```

Figura 3.6: Certificado SSL

3.2.2. Almacenar informacion sensible

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.



Figura 3.7: Contraseñas en texto plano

Solución

Para solucionar este problema de la mejor forma posible debemos tener tres puntos bien definidos:

1. Configurar el factor de costo apropiado
 - El factor de costo (work factor) en bcrypt determina el número de iteraciones utilizadas en el cálculo del hash. Un valor mayor implica una contraseña más segura, pero también requiere más tiempo para calcular el hash. Un valor razonable es 12 o más, dependiendo del hardware y las necesidades de rendimiento.
2. Usar un algoritmo de cifrado seguro.
 - En nuestro caso usaremos BCrypt. CRYPT_BLOWFISH se usa para crear el hash. Producirá un hash estándar compatible con crypt() utilizando el identificador "\$2y\$". El resultado siempre será un string de 60 caracteres, o false en caso de error.
3. Generar un semilla aleatoria para cada usuario.

- BCrypt ya gestiona las semillas de forma automática y en el manual de PHP no recomiendan su uso de otra manera. Más información en <https://www.php.net/manual/es/function.password-hash.php>

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

Figura 3.8: Contraseñas encriptadas

3.2.4. Configuración errónea de las Cookies

Descripción

La configuración errónea de las cookies se refiere a la práctica de establecer parámetros o atributos de las cookies de manera inadecuada, lo que puede tener consecuencias negativas en términos de seguridad, privacidad y funcionalidad en una aplicación web.

Solución

Para solucionar este problema, hemos añadido una configuración segura a las cookies de nuestro sitio web.

```
<?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 3.9: Configuración de las cookies

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 3.10: Parametrizar consulta SQL

3.4. Diseño inseguro

3.4.1. Auditorias de seguridad

Descripción

Una auditoría de seguridad es un proceso sistemático de evaluación y revisión de sistemas, redes, aplicaciones o entornos tecnológicos con el propósito de identificar vulnerabilidades, debilidades y riesgos de seguridad. El objetivo principal de una auditoría de seguridad es garantizar que los controles de seguridad estén implementados adecuadamente y cumplan con los estándares de seguridad, y proporcionar recomendaciones para mejorar la protección de activos y datos contra amenazas y ataques cibernéticos.

Solución

La solución a este problema es realizar una auditoría de seguridad de forma periódica para detectar posibles fallos de seguridad.

3.4.2. Reutilizacion de codigos seguros

Descripción

La reutilización de código seguro se refiere a la práctica de aprovechar componentes de software previamente probados y seguros en lugar de escribir código desde cero. Esto no solo puede acelerar el desarrollo de software, sino que también puede reducir el riesgo de introducir vulnerabilidades de seguridad.

Solución

En nuestro proyecto se han implementado estas practicas mediante la reutilización del archivo `validar.js` en el que se encuentran las funciones de validación de todos los formularios de nuestro sitio web.

3.5. Configuración de seguridad insuficiente

3.5.1. Entornos de desarrollo

Descripción

GitHub

Solución

3.5.2. Plataforma minima

Descripción

Solución

3.5.3. Despliegue seguro

Descripción

Es importante que a la hora de montar nuestro servidor web no haya archivos que puedan ser accesibles desde el exterior y que puedan contener información sensible como contraseñas. Esto a nosotros nos ocurre en el `docker-compose.yml`

Solución

Hemos creado un `.env` donde se almacenan las contraseñas y demás información sensible y hemos añadido el archivo al `.gitignore` para que no se suba al repositorio.

3.5.4. Cabeceras CSP

Descripción

Las cabeceras CSP son una medida de seguridad utilizada en la programación web para mitigar los riesgos asociados con ataques de Cross-Site Scripting (XSS) y otros tipos de ataques de inyección de código malicioso en páginas web.

Solución

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

3.5.5. Cabeceras HSTS

Descripción

Las cabeceras HSTS (HTTP Strict Transport Security) son una medida de seguridad utilizada en la programación web para garantizar que las comunicaciones entre un navegador web y un sitio web se realicen a través de una conexión segura y encriptada utilizando el protocolo HTTPS (HTTP Secure). HSTS ayuda a prevenir ataques de tipo man-in-the-middle (MitM) que podrían exponer datos sensibles o comprometer la seguridad de la comunicación.

Solución

Para solucionar este problema, hemos añadido una cabecera 'Strict-Transport-Security' con los siguientes valores:

3.5.6. Cabeceras X-XSS-Protection

Descripción

Las cabeceras "X-XSS-Protection" son una medida de seguridad utilizada en la programación web para ayudar a prevenir ataques de tipo Cross-Site Scripting (XSS). Los ataques XSS ocurren cuando un atacante inyecta código JavaScript malicioso en una página web, que luego se ejecuta en el navegador de un usuario sin su conocimiento. Estas cabeceras se utilizan para controlar y mitigar estos ataques.

Solución

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

3.5.7. Cabeceras X-Content-Type-Options

Descripción

Las cabeceras "X-Content-Type-Options" son una medida de seguridad utilizada en la programación web para mitigar ciertos tipos de ataques, como ataques de tipo MIME-sniffing. Estas cabeceras se utilizan para controlar cómo el navegador web interpreta y muestra el contenido de una página web.

Solución

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

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

Figura 3.11: Cabecera X-Content-Type-Options

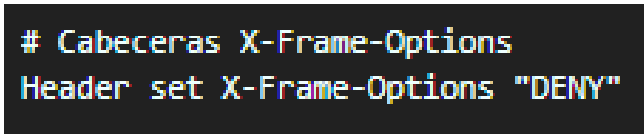
3.5.8. Cabeceras X-Frame-Options

Descripción

Las cabeceras "X-Frame-Options" son una medida de seguridad utilizada en la programación web para controlar si una página web puede ser incrustada o mostrada dentro de un marco (frame) de otro sitio web. Estas cabeceras se utilizan para prevenir ataques de clickjacking, en los cuales un atacante puede ocultar contenido malicioso detrás de una página legítima y engañar a los usuarios para que hagan clic en elementos sin su consentimiento.

Solución

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



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

Figura 3.12: Cabecera X-Frame-Options

3.5.9. Cabeceras Cache-Control

Descripción

Las cabeceras 'Cache-Control' son utilizadas en las respuestas HTTP enviadas por un servidor web para controlar cómo los recursos web deben ser almacenados en la memoria caché del navegador o en servidores intermedios (como proxies) y cómo se deben comportar en términos de almacenamiento y actualización. Estas cabeceras son fundamentales para gestionar la eficiencia de la carga de páginas web, la seguridad y la privacidad.

Solución

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

```
<!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 3.13: Cabecera Cache-Control

3.5.10. Configuración PHP

Descripción

Cuando usamos PHP en un servidor web es importante el configurarlo de forma segura para evitar que se filtre información que pueda ayudar a los atacantes a encontrar una vulnerabilidad en nuestro sistema.

Solución

Hemos ocultado la versión de PHP en las peticiones a nuestro servidor para evitar que un atacante pueda usar esta información para encontrar una vulnerabilidad en nuestro sistema. Para ello hemos creado el `php.ini`

```
; Decides whether PHP may expose the fact that it is installed on the server
; (e.g. by adding its signature to the Web server header). It is no security
; threat in any way, but it makes it possible to determine whether you use PHP
; on your server or not.
; https://php.net/expose-php
expose_php = Off
```

Figura 3.14: Ocultar versión de PHP

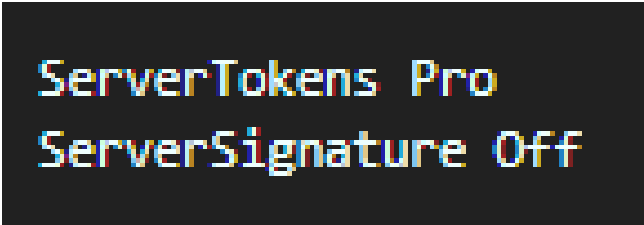
3.5.11. Configuración Apache

Descripción

Al igual que con PHP en el punto anterior es importante configurar Apache para que muestre la mínima información al exterior.

Solución

Para ello, al igual que con PHP, hemos ocultado las versiones del servidor para evitar en la medida de lo posible que el atacante encuentre vulnerabilidad en nuestro servidor. Para ello hemos editado el `apache2.conf`

A screenshot of a text editor showing two lines of configuration code. The first line is `ServerTokens Pro` and the second line is `ServerSignature Off`. The text is in a light blue, monospaced font on a dark background.

```
ServerTokens Pro
ServerSignature Off
```

Figura 3.15: Ocultar versión de Apache

3.6. Componentes vulnerables y obsoletos

3.6.1. Control de versiones de los componentes

Descripción

Es crítico mantener actualizados los sistemas operativos y el software de seguridad con las últimas actualizaciones y parches 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. Autenticacion de dos factores

3.7.2. Contraseñas debiles o por defecto

3.7.3. Invalidacion de sesiones

Descripción

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. Configuracion de logs del sistema

3.9.2. Implementacion de un log

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

3.10.1. Control del trafico

Wireshark/tcpdump

3.10.2. Validacion de accesos

Snort

3.10.3. Asegurar el código

GitHub

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

4.1. ZAP

4.2. sqlmap

4.3. nmap

4.4. Metasploit

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>