eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Sistema Web: Los Montamarcos

Sistemas de Gestión de Seguridad de Sistemas de Información

Laura Calvo

Marco Lartategui

Iker Fuente

Tabata Morente

Iván Herrera

Gorka Bidaguren

07/11/2025

Ingeniería Informática de Gestión y Sistemas de Información

ÍNDICE

1. INTRODUCCIÓN..... 2
2. DEBILIDADES SOLUCIONADAS.....3
    2.1 Broken Access Control (CWE: 284 y WASC: 2).....3
    2.2 Cross-Site Request Forgery (CSRF) (CWE: 352 y WASC: 9).....6
    2.3 SQL Injection - MySQL (CWE: 89 y WASC: 19).....7
    2.4 Cabecera Content Security Policy (CSP) no configurada (CWE: 693 WASC: 15) 8
    2.5 Falta de cabecera Anti-Clickjacking (CWE ID: 1021 WASC ID: 15).....9
    2.6 El servidor filtra información de versión a través del campo "Server" del encabezado de respuesta HTTP..... 10
    2.7 El servidor divulga información mediante un campo(s) de encabezado de respuesta HTTP ""X-Powered-By"" (CWE ID: 200 WASC ID: 13).....10
    2.8 Falta encabezado X-Content-Type-Options (CWE: 693 y WASC: 19)..... 11
    2.9 Guardar contraseñas en texto plano.....12
    2.10 Hardcoding de credenciales (CWE: 798)..... 13
3. ANÁLISIS DE ZAP..... 15
4. CONCLUSIÓN.....17
5. BIBLIOGRAFÍA..... 19

1. INTRODUCCIÓN

Tras la auditoría de seguridad realizada en la entrega 2 sobre la página web “Los Montamarcos”, se nos ha pedido desarrollar una nueva versión del sistema web que corrija las vulnerabilidades detectadas en la entrega anterior. Para ello, se ha creado una nueva rama en GitHub denominada “entrega_3”, en la cual se implementan las modificaciones necesarias para eliminar o corregir dichas debilidades.

Después de haber realizado varias pruebas y escaneos, llegamos a la conclusión de que nuestra página web contiene nueve fallos de seguridad notables. Estas alertas notificadas por el proxy dejan en evidencia que nuestra página web, como estaba previsto que sucediera, no disponía de un nivel de seguridad aceptable, teniendo incluso varias alertas que comprometen enormemente la seguridad de los datos más importantes de la página web.

Es por ello, que durante la realización de la entrega hemos dado con una solución a los nueve fallos de seguridad más importantes que se encontraron mediante los análisis. De esta forma, podemos certificar una seguridad más estable tanto a los usuarios que accedan al sistema como al código de la web en sí.

A continuación explicaremos las mejoras realizadas en el sistema web, indicando el método empleado para solucionar cada vulnerabilidad detectada en la entrega anterior. De este modo se busca no solo reforzar la seguridad del sistema, sino también mejorar la calidad y la seguridad del código frente a posibles amenazas o ataques futuros.

2. DEBILIDADES SOLUCIONADAS

2.1 Broken Access Control (CWE: 284 y WASC: 2)

El problema de *Broken Access Control* permite acceder a páginas directamente mediante la URL sin comprobar que el usuario haya iniciado sesión o que tiene permiso para entrar a dicha URL.

Para arreglar este error, hay que guardar los datos en variables de sesión en el archivo login.php y proteger las páginas privadas para verificar que solo entran los usuarios.

Para ello, se ha creado **auth.php**, donde:

Primero, se tiene que crear/reanudar una sesión para el usuario actual, que además permite utilizar variables de sesión para almacenar los datos del usuario autenticado:

```
session_start(); //inicia sesión para poder guardar los datos
```

Regeneramos el ID del usuario para poder prevenir el secuestro de sesión:

```
// Regenerar ID de usuario cada 5 minutos para evitar secuestro de
sesión
if (!isset($_SESSION['creada'])) {
   session_regenerate_id(true);
   $_SESSION['creada'] = time();
} elseif (time() - $_SESSION['creada'] > 300) { // cada 5 minutos
   session_regenerate_id(true);
   $_SESSION['creada'] = time();
```

Luego, se tiene que verificar si hay ya una sesión activa. Si no existe, significa que el usuario no ha iniciado sesión, entonces lo redirige a la página de inicio con un mensaje de error:

```
//Verificar si hay sesion activa
      if(!isset($_SESSION['email']))
      {
         header("Location: /index.html?error=acceso_no_autorizado");
         exit;
      }
```

Se tiene que verificar que el parámetro que se consigue con GET coincida con la sesión actual, y si no coincide, lanza un aviso de que no tiene permiso para acceder a la página:

```
//Verificar que hay parametro GET y que coincida con la sesion
   if (!isset($_GET['user']) || $_GET['user'] !== $_SESSION['email']) {
       echo "No tienes permiso para acceder a esta página.";
       exit;
       }
```

Además, se controla el tiempo de inactividad en la web. Si el usuario supera el tiempo de inactividad de 30 minutos se elimina la sesión y le redirige a la página de inicio. En caso de que no haya pasado ese tiempo, se actualiza la variable de último acceso para poder mantener la sesión activa:

```
$inactividad_max = 60*30;
if (isset($_SESSION['ultimo_acceso'])) {
   if (time() - $_SESSION['ultimo_acceso'] > $inactividad_max) {
       session_unset();
       session_destroy();
       header("Location: /index.html?error=session_expirada");
       exit;
   }
}

$_SESSION['ultimo_acceso'] = time();
```

En el **login.php** , para mantener la sesión activa durante los 30 minutos, es necesario configurar los valores del tiempo que dura la cookie de sesión en el navegador del usuario y el tiempo que el servidor mantiene la sesión almacenada antes de eliminarla. y luego iniciamos la sesión:

```
//Mantener sesion activa durante 30 minutos
ini_set('session.cookie_lifetime',60*30);
ini_set('session.gc_maxlifetime',60*30);

session_start();
```

Asimismo, para conseguir una sesión segura se regenerará el id de sesión:

```
session_regenerate_id(true);
```

En **pagUsuario.php** primero se incluye auth.php para comprobar la sesión activa, que la sesión no ha expirado… Luego utiliza cabeceras para evitar caché del navegador, y así en consecuencia evitar que si el usuario cierra sesión y vuelve para atrás con el botón del navegador, pueda ver datos privados, lo que previene la exposición de información sensible a través de la caché:

```
include('../auth.php');
// Evitar caché del navegador
header("Cache-Control: no-store, no-cache, must-revalidate,
max-age=0");
header("Cache-Control: post-check=0, pre-check=0", false);
header("Pragma: no-cache");
header("Expires: 0");
```

Asimismo, a la hora de cerrar sesión en la web, para que se elimine toda la información del cliente (la cookie) y del servidor y evitar que otro usuario utilice una sesión antigua siga teniendo acceso:

```
<?php
session_start();

// Destruir todos los datos de sesión
session_unset();     // Limpia las variables de sesión
session_destroy();   // Elimina la sesión en el servidor

// Borrar la cookie de sesión
if (ini_get("session.use_cookies")) {
   $params = session_get_cookie_params();
   setcookie(
       session_name(),
       '',
       time() - 42000,
       $params["path"],
       $params["domain"],
       $params["secure"],
       $params["httponly"]
   );
}

// Redirigir al usuario a la página principal
header("Location: /index.html?logout=ok");
exit;
?>
```

2.2 Cross-Site Request Forgery (CSRF) (CWE: 352 y WASC: 9)

Para solucionar este problema de seguridad se crearán Tokens CSRF (claves aleatorias únicas generadas por el servidor) que se almacenarán en la sesión del usuario. Se genera en **auth.php**, creando un identificador único y seguro para cada sesión:

```
// Generar token CSRF si no existe
if (!isset($_SESSION['csrf_token'])) {
   $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
```

En el **index.php** de login se genera el token CSRF si no existe y luego dentro del html, se inserta un campo oculto que viaja junto al formulario que el usuario envía desde su ordenador:

```
session_start();

// Generar el token CSRF si no existe
if (!isset($_SESSION['csrf_token'])) {
 $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
?>

<!DOCTYPE html>
...
<!-- Token CSRF -->
                <input   type="hidden"   name="csrf_token"   value="<?=
htmlspecialchars($_SESSION['csrf_token']) ?>">

   </form>
```

Además, en login.php se inicia la sesión y se verifica el token CSRF:

```
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    if (!isset($_POST['csrf_token']) || !isset($_SESSION['csrf_token'])
|| $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
        // Token inválido o ausente
        header("Location: index.php?error=csrf");
        exit;
   }
} else {
   echo "Método no permitido.";
   exit;}
```

En el index.php de add_item y en el de delete_item,y el index.html de modify_user (ya que utilizan POST), se inserta un campo oculto que viaja junto al formulario que el usuario envía desde su ordenador (Teniendo en cuenta de que en los index.php incluimos al principio auth.php para verificar la autenticación del usuario antes de mostrar cualquier contenido):

```
<?php
echo "<input type='hidden' name='csrf_token' value='" .
htmlspecialchars($_SESSION["csrf_token"]) . "' />";
    ?>
```

En **modify_datos.php**, **procesar_adicion.php** y en **procesar_borrado.php**, justo al principio se verifica si el token coincide con el guardado en la sesión, si no coincide la solicitud no será válida y aparecerá el mensaje correspondiente junto a un botón para volver a la página de inicio:

```
session_start();
//  Verificar token CSRF
if ($_SERVER["REQUEST_METHOD"] === "POST") {
      if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !==
$_SESSION['csrf_token']) {
     echo "❌ Solicitud no válida (token CSRF incorrecto o ausente).";
     echo "<div class='volver-container'>";
     echo "<a href='../index.html'>Volver</a>";
     echo "</div>";
     exit;
  }
}
```

2.3 SQL Injection - MySQL (CWE: 89 y WASC: 19)

La inyección SQL se basa en poner comandos SQL en los textbox de los formularios. Para poder arreglarlo debemos verificar el tipo de dato antes de ejecutar la consulta y ejecutar consultas preparadas. Las consultas preparadas separan la sintaxis SQL de los datos, de esta manera si se han añadido comandos SQL en el formulario, se tratarán como datos y no se ejecutarán.

Validaciones:

```
if ($email === '' || !filter_var($email, FILTER_VALIDATE_EMAIL)) {
   $errors[] = "Email no válido.";
}
```

Valida que el email tiene un formato correcto para que no se pueda introducir nada raro (FILTER_VALIDATE_EMAIL es una función de php que verifica el formato de un correo electrónico).

Consultas preparadas:

```
$stmt_apuesta = $conn->prepare("INSERT INTO apuesta (cerdo, idCarrera,
idUs, cantidad) VALUES (?, ?, ?, ?)");
$stmt_apuesta->bind_param("ssii", $cerdo, $carrera, $idUs, $cantidad);
```

Los datos se colocan con un signo “?” y se introducen más adelante, así solo se consideran como datos y se ejecuta con un:

```
$stmt_apuesta->execute()
```

2.4 Cabecera Content Security Policy (CSP) no configurada (CWE: 693 WASC: 15)

La cabecera Content Security Policy es esencial para cualquier página web, certificando una mayor seguridad ante ataques de inserción de código. Es por ello, que al no tenerla en nuestra web, suponía un gran riesgo.
Para aliviar el riesgo, es tan simple como introducir la configuración de la cabecera en un nuevo archivo de configuración para todas las cabeceras: security-headers.conf. La configuración que se ha introducido es la siguiente:

```
# security-headers.conf
<IfModule mod_headers.c>
   # Content Security Policy
   Header always set Content-Security-Policy "default-src 'self';
script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self'
data:; font-src 'self'; connect-src 'self'; media-src 'self';
object-src 'none'; base-uri 'self'; frame-ancestors 'none'; form-action
'self';"
</IfModule>
```

Las opciones de la cabecera usadas se explicarán a continuación:
default-src 'self': permite cargar recursos solo si son del mismo dominio.
script-src 'self': permite cargar y ejecutar scripts solo si son del mismo dominio.
style-src 'self' 'unsafe-inline': permite cargar código css en línea, por ejemplo, <style>. Si bien esto puede suponer un riesgo, es necesario en este sistema ya que se hace varios usos de <style> en los html. Este riesgo está ligado a la inyección de código, lo cual nos encargamos de evitar en otros apartados.
img-src 'self' data: permite cargar imágenes solo si son del mismo dominio.
font-src 'self': permite cargar fuentes de texto solo si son del mismo dominio.
connect-src 'self': permiten hacer peticiones AJAX, fetch etc. solo si se hacen al dominio.
media-src 'self': permite que se carguen audios y videos solo desde el mismo dominio.
object-src 'none': bloquea el uso de objetos embebidos antiguos.
base-uri 'self': permite que la etiqueta <base> provenga solo del mismo dominio.
frame-ancestors 'none': bloquea que otros sitios puedan insertar la web en un <iframe>, evitando clickjacking.
form-action 'self': permite que se puedan enviar formularios solo a URLs del mismo dominio.

Para que esta configuración se aplique, es necesario que el propio dockerfile se encargue de copiarlas a la configuración apache del contenedor para luego ponerlas en marcha con a2enmod headers y a2enmod security-headers.

```
COPY config/security-headers.conf
/etc/apache2/conf-available/security-headers.conf
RUN a2enmod headers && a2enconf security-headers
```

Por último, para comprobar que cualquier cabecera está configurada correctamente, se puede inspeccionar la petición en el navegador. Una vez dentro de la página web, se debe pulsar *f12*, entrar en la pestaña de red o network y pulsar sobre la petición correspondiente. Al hacerlo, en el apartado de cabeceras de respuesta, debería de aparecer la cabecera content-security-policy junto al resto de cabeceras.

2.5 Falta de cabecera Anti-Clickjacking (CWE ID: 1021 WASC ID: 15)

El clickjacking consiste en insertar la página web en un iframe de otra página web, pudiendo engañar al usuario haciéndole pulsar botones que no pertenecen a la página web legítima. Para solucionar esto, se deben añadir ciertas cabeceras que eviten que la web se añada en un iframe de sitios ajenos al sistema. En cierto sentido, esta funcionalidad ya fue añadida en la cabecera content-security-policy del apartado anterior, en específico, la opción frame-ancestors 'none'. Pero además de esta, es conveniente añadir otra cabecera ya fuera de la de CSP que se encarga de la misma función:

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

Esta cabecera se asegura de que no se pueda incrustar la página web en ningún iframe de ningún sitio. Si bien es recomendable añadirla también, en general, frame-ancestors cumple mejor con su trabajo ya que cuenta con más control y da soporte a más navegadores modernos.

En conclusión, para evitar clickjacking lo más recomendable es implementar las dos para una mayor seguridad y compatibilidad.

2.6 El servidor filtra información de versión a través del campo "Server" del encabezado de respuesta HTTP

Se ha añadido un nuevo archivo llamado "security.conf" para poder ocultar la versión que utilizamos del servidor Apache y que el atacante no pueda conseguir información para buscar un ataque. El archivo tiene el siguiente código:

```
ServerSignature Off
ServerTokens Prod
```

Además en el archivo "docker-compose.yml" se ha montado el anterior archivo:

```
- ./config/security.conf:/etc/apache2/conf-enabled/security.conf
```

2.7 El servidor divulga información mediante un campo(s) de encabezado de respuesta HTTP ""X-Powered-By"" (CWE ID: 200 WASC ID: 13)

Se ha añadido un nuevo archivo llamado “security.ini” para poder ocultar la versión que utilizamos de php y que el atacante no pueda conseguir información para buscar un ataque. El archivo tiene el siguiente código:

```
 expose_php=Off
```

Además en el archivo “docker-compose.yml” se ha montado el anterior archivo:

```
- ./config/security.ini:/usr/local/etc/php/conf.d/security.ini
```

2.8 Falta encabezado X-Content-Type-Options (CWE: 693 y WASC: 19)

El encabezado X-Content-Type-Options: nosniff indica a los navegadores que no intenten adivinar el tipo de contenido de los archivos servidos por el servidor. Esto evita que scripts maliciosos o archivos HTML/JS se ejecuten si el navegador interpreta erróneamente el tipo de archivo.
Para ello de ha escrito lo siguiente en el archivo security-headers.conf de la carpeta config:

```
<IfModule mod_headers.c>
   # Evita MIME sniffing
   Header always set X-Content-Type-Options "nosniff"
</IfModule>
```

Pero como esto solo funciona cuando mod_headers de apache está activo se ha tenido que crear un script start-web.sh para activar esto al iniciar docker.

Este script se llama desde docker-compose.yml de la siguiente manera:

```
web:
  image: web
```

```
  environment:
    - ALLOW_OVERRIDE=true
    - HOSTNAME=db
    - USER=admin
    - PASSWORD=test
    - DB=database
  ports:
    - "81:80"
  links:
    - db
  volumes:
    - ./app:/var/www/html/
    - ./config/security.ini:/usr/local/etc/php/conf.d/security.ini
    - ./config/security.conf:/etc/apache2/conf-enabled/security.conf
    - ./start-web.sh:/start-web.sh   # Montar el script
  command: ["/start-web.sh"]  # Ejecutar el script al arrancar
```

2.9 Guardar contraseñas en texto plano

Una de las brechas de seguridad de nuestra web era la de guardar las contraseñas en texto plano en la base de datos, por lo que si alguien no autorizado obtuviese acceso a dicha base de datos todas las cuentas de los usuarios se verían comprometidas.

Para ello se ha implementado una función que convierte la contraseña introducida al registrarte en un hash con salt para evitar colisiones si existen dos contraseñas iguales.

```
// --- Crear el hash de la contraseña ---
  $hashedPassword = password_hash($contrasena, PASSWORD_DEFAULT);
  if ($hashedPassword === false) {
      throw new Exception("No se pudo generar el hash de la
                                                  contraseña.");
  }
```

Luego para poder comprobar si la contraseña introducida es la correcta hay que convertirla a hash y compararla con la guardada en la base de datos y si coinciden entonces la contraseña es la correcta.

```
  // Verificar la contraseña con hash
  if (password_verify($contrasena, $row["contrasena"])) {
      // Redirigir al usuario con su email en la URL
      header("Location: /pagUsuario/pagUsuario.php?user=" .
```

```
                                                urlencode($row["email"]));
        exit;
    } else {
        header("Location: index.html?error=2"); // contraseña incorrecta
        exit;
    }
```

Además, las contraseñas que están dentro del archivo `database.sql` de los usuarios `mikel@example.com` y `aitor@example.com` también se les ha aplicado el algoritmo resumen.

```
-- Ejemplo comentado para insertar usuarios
INSERT INTO `usuarios` (`id`, `nombre`, `contrasena`, `dni`,
`telefono`, `fecha`, `email`) VALUES
(1, 'mikel',
'$2y$10$6VtxvE3BNc4wj/qzYohte.KVncRVLzGT5Wx9QNKivU7rscyc/4K1C',
'12345678A', 600123456, '2025-10-01', 'mikel@example.com'),
(2, 'aitor',
'$2y$10$V5FwQjcIUZ/SXb2uk1aCVOsC6gnUX.BHWx7RKaljPiTqeXJ8lB5aa',
'87654321B', 611987654, '2025-10-05', 'aitor@example.com');
```

2.10 Hardcoding de credenciales (CWE: 798)

El hardcoding de credenciales consiste en tener escrito de forma explícita información sensible, como claves, tokens y contraseñas, en el propio código fuente. El mayor peligro que causa esta práctica es exponer dicha información accesible en el caso de que una persona ajena al sistema obtuviera el código fuente. Para ello, se ha propuesto la siguiente solución:
Consiste en tener variables de entorno que guardan las credenciales de la base de datos. Para ello, se ha modificado el archivo `docker-compose.yml` para que el contenedor web tenga dichas variables. Aquí se mostrarán los cambios.

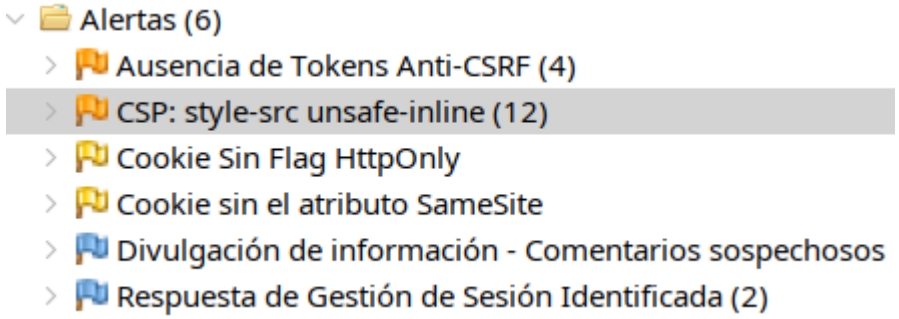
```
web:
    image: web
    environment:
      - ALLOW_OVERRIDE=true
      - HOSTNAME=db
```

```
      - USER=admin
      - PASSWORD=test
      - DB=database
    ports:
      ...
    links:
      ...
    volumes:
      ...
db:
  ...
phpmyadmin:
  ...
```

Entonces, siempre que se vaya a conectar con la base de datos habrá que utilizar los valores guardados en las variables de entorno. El siguiente código es un ejemplo de los pasos a realizar:

```
<?php
        ...

        $hostname = getenv("HOSTNAME");
        $username = getenv("USER");
        $password = getenv("PASSWORD");
        $db = getenv("DB");
        $conn =
mysqli_connect($hostname,$username,$password,$db);

        if ($conn->connect_error) {
                die("Database connection failed: " .
$conn->connect_error);
        }

        ...

        $conn->close();
?>
```

3. ANÁLISIS DE ZAP

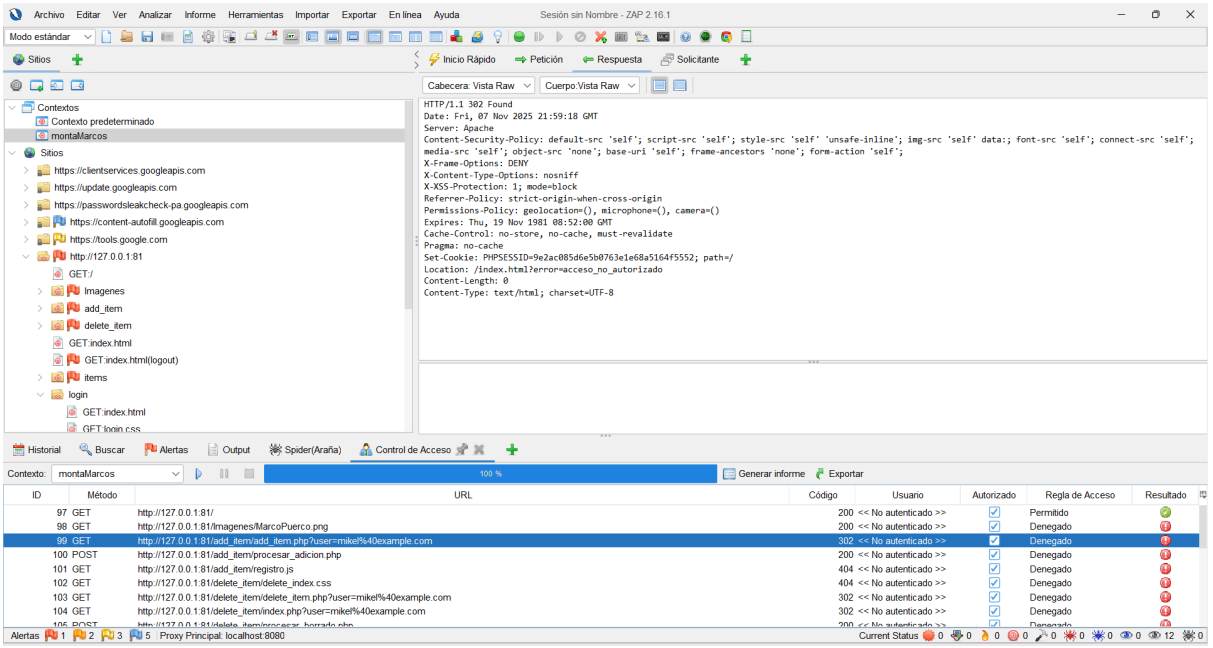


CSP: style-src unsafe-inline: Como se ha explicado en el apartado 2.3, al usar la opción style-src 'self' 'unsafe-inline' en la cabecera CSP, se genera un riesgo que el proxy ZAP notifica. Sin embargo, este riesgo se limita a la inyección de código en CSS, por lo tanto no supone un riesgo para la estructura general de la página web ni para los datos, volviéndo un riesgo muy leve.

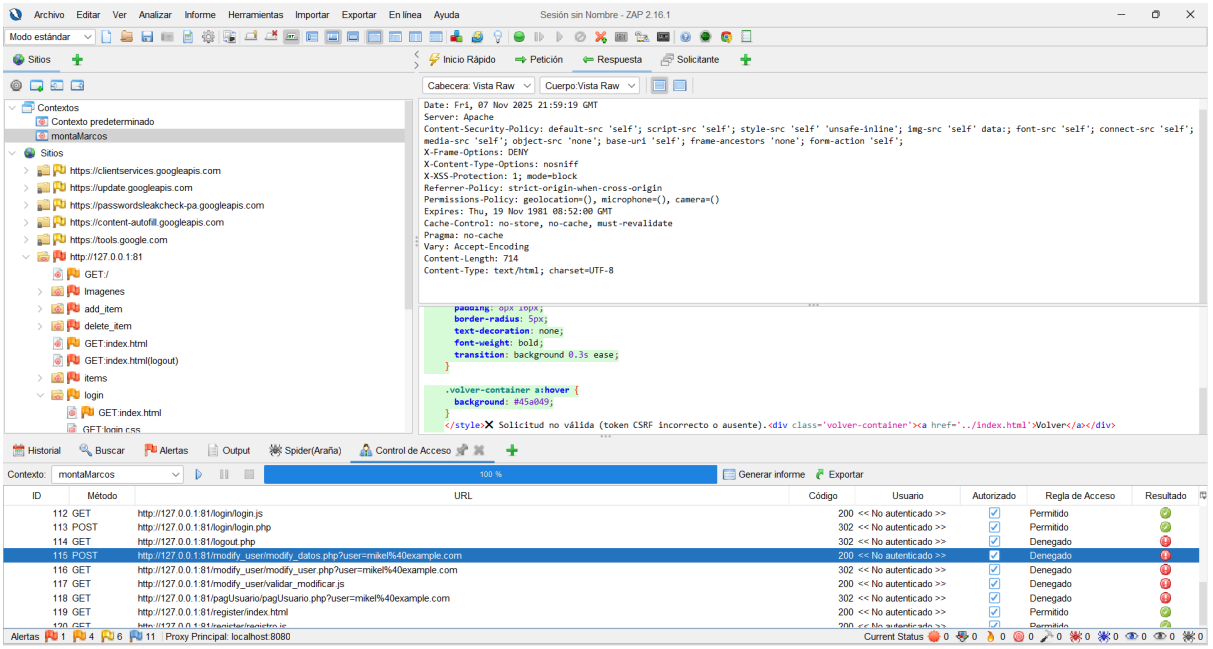
		<< No autenticado >>		mikel	
Método	URL	¿Autorizado?	Control de Acceso	¿Autorizado?	Control de Acceso
GET	http://127.0.0.1:81/login/login.js	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/login/index.html?error=1	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/sitemap.xml	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/Imagenes	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/logout.php	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/Imagenes/	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/items/items.php?user=mikel%40example.com	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/delete_item/delete_index.css	Sí	Inválido	Sí	Válido
POST	http://127.0.0.1:81/delete_item/procesar_borrado.php?user=$email	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/modify_user	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/items/show_items.php?user=mikel%40example.com	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/add_item	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/add_item/procesar_adicion.php	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/modify_user/validar_modificar.js	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/modify_user/modify_datos.php?user=mikel%40example.com	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/add_item/add_item.php?user=mikel%40example.com	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/add_item/	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/modify_user/modify_user.php?user=mikel%40example.com	Sí	Inválido	Sí	Válido
POST	http://127.0.0.1:81/register/procesar_registro.php	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/register/index.html	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/delete_item/delete_item.php?user=mikel%40example.com	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/items/	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/login/login.css	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/Imagenes/MarcoPuerco.png	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/register/registro.js	Sí	Válido	Sí	Válido

GET	http://127.0.0.1:81/	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/robots.txt	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/login/index.html	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/login	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/login/login.php	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/pagUsuario/pagUsuario.php?user=mikel%40example.com	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/delete_item	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/pagUsuario/	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/delete_item/	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/login/	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/index.html	Sí	Válido	Sí	Válido
GET	http://127.0.0.1:81/index.html?logout=ok	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/pagUsuario	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/items	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/modify_user/	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/delete_item/procesar_borrado.php	Sí	Inválido	Sí	Válido
GET	http://127.0.0.1:81/add_item/registro.js	Sí	Inválido	Sí	Válido

A pesar de que ZAP detecte como si se hubiera podido entrar como un usuario sin credenciales a una página donde requiere haberse identificado, la página web no permite dicho acceso en realidad. La razón de esto es porque se comprueba si existe el token y si este corresponde al usuario que corresponde a la página. En caso contrario, se mostraría un mensaje de que no tiene permiso de acceder a dicha web. Sin embargo, el problema está en que no sé responde con el estado HTTP 403 “Forbidden”, que representa el prohibido, sino que el sitio web responde con el estado 200 “OK”. Las siguientes imágenes muestran el ejemplo de respuesta de http cuando se intenta acceder a un enlace sin las credenciales correspondientes.
Por un lado, la cabecera de la respuesta tiene el estado 302 “Found”, dándonos a entender que redirige al usuario a la página de la url a la derecha de “Location:”, que en este caso es la página principal del sitio. Entonces, en este caso el usuario no es capaz de acceder a las páginas de usuario.

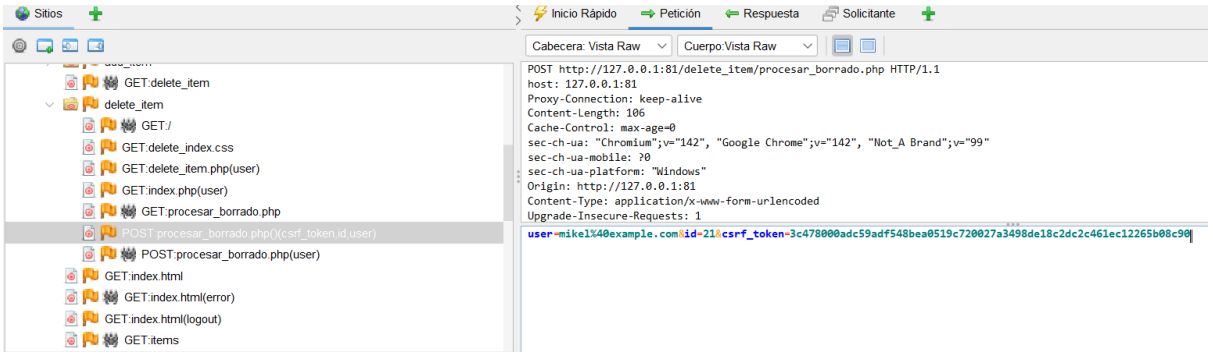


Otra manera en la que el mensaje protege a los usuarios es tal que así. La respuesta en el protocolo http de la web será de 200 OK. Pero, el cuerpo de la respuesta no contiene información privada de algún usuario, en cambio, contiene mensajes similares a los siguientes:
`Solicitud no válida (token CSRF incorrecto o ausente).`
Abajo habrá una imagen de ejemplo.



Por tanto, aunque la extensión de ZAP no ha servido de ayuda para comprobar el funcionamiento, se sigue pudiendo ver de forma manual las cabeceras de respuestas. Así, se demuestra que la página web está segura de la alerta Broken Access Control.

Además, también nos han quedado tres alertas en `procesar_borrado.php`. Sin embargo, consideramos que la alerta no debería saltar, ya que el script utiliza métodos para comprobar la existencia del token. Además en ZAP se puede ver que realiza la petición con el token. Se mostrará imagen de lo siguiente. También se enseñarán las alertas que han quedado por resolver.



```php
//  Verificar token CSRF
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
        echo "❌ Solicitud no válida (token CSRF incorrecto o ausente).";
        echo "<div class='volver-container'>";
        echo "<a href='../index.html'>Volver</a>";
        echo "</div>";
        exit;
    }
```

```
}
```

4. CONCLUSIÓN

Una vez aplicadas las correcciones y ejecutado nuevamente ZAP sobre la versión actualizada del sistema web “Los Montamarcos”, se ha comprobado que las vulnerabilidades detectadas en la entrega anterior han sido corregidas y solucionadas correctamente. Los cambios implementados han permitido fortalecer aspectos clave del sistema, tales como el control de acceso, la gestión de sesiones, el tratamiento de entradas de usuario y la configuración de cabeceras de seguridad.

Este proceso de corrección nos ha ayudado a entender la importancia de mantener una revisión continua de la seguridad en el desarrollo web, ya que las amenazas evolucionan con el tiempo y pueden surgir nuevas debilidades incluso en sistemas previamente auditados. Por eso, es importante seguir buenas prácticas, mantener las actualizaciones al día y hacer pruebas periódicas de seguridad.

En conclusión, el sistema web ha alcanzado un nivel de seguridad más sólido y estable, cumpliendo los objetivos propuestos para esta entrega y sentando una base más segura para futuras mejoras o ampliaciones en el proyecto.

5. BIBLIOGRAFÍA

- **Arun Testing.** (2023, 21 mayo). Authenticated Scan Using OWASP ZAP Form based authentication [Video]. Youtube. https://www.youtube.com/watch?v=e-ry_Poct2o=191s&t
- **PHP.net.** (s.f.). password_hash — Creates a password hash. https://www.php.net/manual/en/function.password-hash.php
- **Stack Overflow.** (2020, 13 febrero). Docker compose gives invalid environment type error. https://stackoverflow.com/questions/60200966/docker-compose-gives-invalid-environment-type-error
- **OWASP Foundation.** (s.f.). OWASP Zed Attack Proxy (ZAP) Project. https://www.zaproxy.org/
- **OpenAI.** (2025). ChatGPT (GPT-5) [Modelo de lenguaje de IA]. https://chat.openai.com/