INFORME SOBRE SOLUCIÓN DE VULNERABILIDADES: SafeFilms

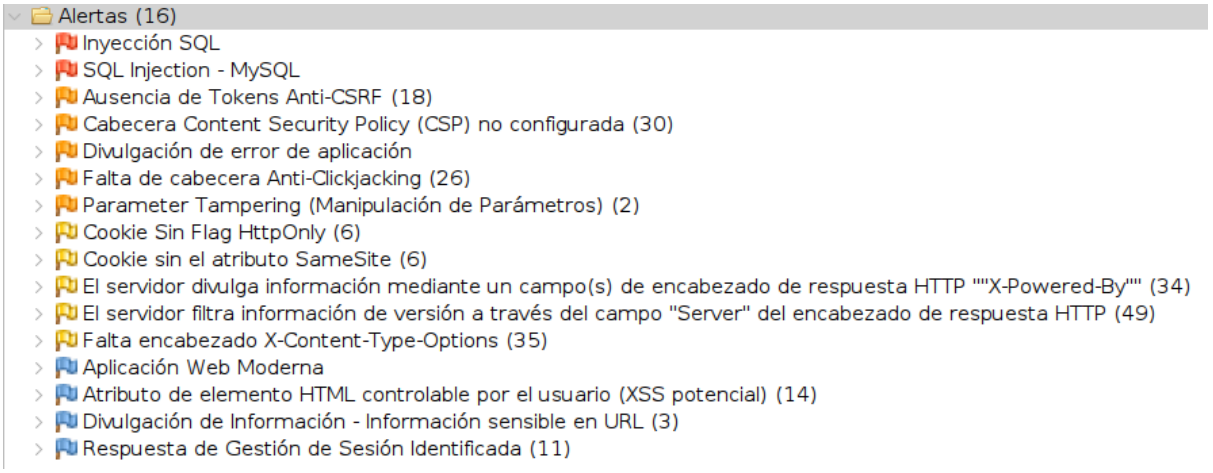
- *Mbarek Galloul Ezzakraoui*
- *Alicia Maizkurrena Moreno*
- *Adrián Vinagre*
- *Maira Gabriela Herbas*
- *Ana Victoria Cernatescu*

***Herramientas**: OWASP ZAP 2.16.1*

ÍNDICE

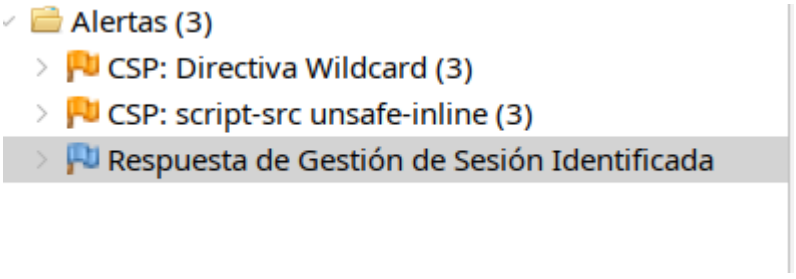
INTRODUCCIÓN

En este informe se explican las modificaciones que se han realizado al código del sistema SafeFilms, con el objetivo de deshacerse de la mayor cantidad de brechas de seguridad posibles. Recapitulando con la anterior entrega, estas son las alertas que se hallaron en la auditoría:



Imágen de primera auditoría, con mayor cantidad de alertas. (Entrega 2)

Tras aplicar los cambios y analizar con OWASP ZAP, estas son las alertas de seguridad que recibe el sistema:



Imágen de la última auditoría de seguridad.

Como se puede observar, se han corregido aquellas relevantes. Las dos primeras alertas se han dejado sin solucionar debido a que su implementación supone una carga de trabajo demasiado grande para esta entrega. Se profundiza en este tema más adelante en el informe.

La última alerta es un aviso sobre que se muestran cookies de sesión mediante PHPSESSID. Hemos configurado la mayoría de atributos de seguridad sobre dicho campo, y la única razón por la que aparece la alerta es porque no tenemos el atributo "$secure" en verdadero, para poder utilizar tanto HTTP como HTTPS.

MODIFICACIONES

INYECCIÓN SQL ELIMINADA

El código original concatenaba directamente las variables, pudiendo un atacante realizar inyecciones de código SQL. Con el código actual ya no es posible, ya que la base de datos recibe la sentencia y los parámetros por separado. Los datos se tratan como valores, no como código SQL. Este problema ocurría principalmente en las páginas *login.php* y *register.php*, que son las páginas donde se conecta con la base de datos. Los cambios realizados han sido:

```
$stmt = $conn->prepare("SELECT idU, contrasena, rol FROM usuarios
WHERE usuario = ?");
   $stmt->bind_param("s", $usuario);
```

Se prepara la consulta en la primera línea y en la segunda se trata el parámetro $usuario como texto literal. Otros cambios que han ayudado a este apartado son: Transformar las contraseñas nuevas que se quieran almacenar en un hash, y verificar que la contraseña coincide con el hash al iniciar sesión.

```
//hashear la contraseña antes de guardarla
      $hash = password_hash($contrasena, PASSWORD_DEFAULT);
```

```
//al iniciar sesión
       if ($result->num_rows > 0 && password_verify($contrasena,
$row['contrasena']))
```

AÑADIDOS TOKENS ANTI-CSRF

Con el anterior código, salta una alerta porque no se comprueba en ningún momento que la petición es enviada intencionalmente por un usuario. Para su resolución se han tenido que añadir tres partes distintas de código:

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

```
// validar el token CSRF
      if  (!isset($_POST['csrf_token'])  ||  $_POST['csrf_token']  !==
$_SESSION['csrf_token']) {
      http_response_code(403);
      die('Error: CSRF token inválido.');
   }
```

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

En la primera parte, se crea el token CSRF si no está ya creado. Esto se aplica a las clases index.php , items.php, y por otro lado también a login.php y register.php porque el usuario puede acceder directamente a estas URLs.

El segundo trozo corresponde a la validación del token. La validación del token es necesaria siempre y cuando el usuario mande un formulario (ya que requiere de una acción POST) e incluso cuando quiera cerrar sesión (acción similar a DELETE).

La última parte corresponde a un fragmento html que se debe añadir siempre y cuando se envíe un formulario que requiera validación.

CABECERA CSP AÑADIDA

Parte de las vulnerabilidades de la página están basadas en la falta de configuración de seguridad en las cabeceras. Para solucionar este problema, se ha añadido al proyecto el archivo ".htaccess". Este tipo de archivo sirve para poder configurar apache en un proyecto concreto, sin necesidad de configurar archivos globales de apache, y sin necesidad de tener acceso a dichos archivos.

En este archivo se ha añadido que la cabecera "Content-Security-Policy" (CSP) debe estar siempre activa (en cualquier página de la web, incluso en páginas de error), y se ha configurado de tal manera que:

- Los archivos de estilo (CSS) solo se pueden cargar desde el orígen de la página web (style-src self)
- Las imágenes solo se pueden cargar desde el orígen de la página web (img-src self)
- Las fuentes solo se pueden cargar desde el orígen de la página web, es decir, deben ser las propias fuentes de la página (http://localhost/login.php, http://localhost/register.php están permitidas pero http://unknown/login.php no) (font-src self)
- Los formularios solo pueden ser enviados a este sitio web (form-action self)
- Solo esta página web puede poner esta misma página web en iframes, es decir, otro sitio será incapaz de hacer accesible esta página web desde sí mismo. (frame-ancestors self)
- Las rutas base, definidas con la etiqueta <base>, solo pueden ser definidas en el orígen de la página web, evitando así que cualquiera pueda definir otro sitio como orígen. (base-uri self)

Hay dos atributos relevantes que no se han tenido en cuenta para la configuración. Éste tema se explica brevemente en la introducción, y aquí se justifican más profundamente las razones de no aplicar los siguientes cambios de configuración al archivo .htaccess:

- Los recursos (scripts, imágenes…) solo se pueden cargar desde el orígen de la página web. (default-src self)
- Los archivos Java Script solo se pueden cargar desde el orígen de la página web (script-src self)

Al crear el sistema web, se introdujo gran parte de la lógica del mismo directamente en los archivos de extensión ".php". Ejemplos de estos casos son los siguientes:

```
<script src='./js/validar_pelicula.js'></script>
<script>
   // Mostrar el diálogo al hacer clic en el botón
   document.querySelector('.btn-add-pelicula').onclick = () => {
     document.querySelector('.add-peli-dialog').showModal();
   };

   // Cerrar el diálogo
   document.querySelector('.btn-cerrar-add-peli').onclick = () => {
     document.querySelector('.add-peli-dialog').close();
   };
 </script>
<?php endif; ?>
 <script src='./js/ver_info_peli.js'></script>
</body>
```

fragmento de código html en items.php, se muestran varias cláusulas "script".

```
<?php
  if (isset($_SESSION['usuario'])) : ?>
    echo "<script>window.location.href='items.php';</script>";
    exit;

  <?php else: ?>
```

fragmento de código html en index.php, se muestran varias cláusulas "script".

```
if ($stmt->execute()) {
    // Éxito: alert + volver al perfil
    echo "<script>
            alert('Datos actualizados correctamente');
            window.location.href='show_user.php?user=".$userId."';
          </script>";
```

fragmento de código de modify_user.php, en el que se usan cláusulas "script" para alertar al usuario.

Y como estos ejemplos, existen más casos en los que, en vez de utilizar un archivo de extensión .js para ejecutar distintas acciones o dar ciertos avisos, se hace directamente (inline) en el código php y html. La configuración "default-scr self y script-src self" precisamente tratan de evitar que el código JavaScript venga de otra parte que no sea un archivo JavaScript perteneciente al orígen del sistema web, con lo que automáticamente inhabilita el código que no siga esa regla.

Este caso es similar al de la configuración "style-src self", ya que sí que se ha invertido tiempo en pasar todas las cláusulas del tipo "style" a un archivo de extensión .css, de tal manera que no haya conflictos con dicha configuración. Cabe resaltar que la carga de trabajo para esta tarea fué significativamente menor y por eso se pudo llevar a cabo.

Además de todo esto, también se modifica el archivo Dockerfile. De esta manera, se modifican los archivos de configuración de apache al levantar el entorno, y se permite la edición de cabeceras:

```
RUN a2enmod headers

# Fuerza que Apache use .htaccess en la carpeta del sitio
RUN sed -i '/<Directory \/var\/www\/>/,/<\/Directory>/ s/AllowOverride
None/AllowOverride All/' /etc/apache2/apache2.conf && \
   sed -i '/<Directory \/var\/www\/html\/>/,/<\/Directory>/
s/AllowOverride None/AllowOverride All/' /etc/apache2/apache2.conf
```

CABECERA ANTI-CLICKJACKING AÑADIDA

El clickjacking puede provocar que un atacante cree botones invisibles, donde un usuario pueda clickar accidentalmente y realizar un acción sin ser consciente. Para solucionar este error, se ha añadido a la cabecera "Content-Security-Policy" frame-ancestors self (mencionado en el anterior apartado) y se ha añadido la cabecera "X-Frame-Options", que realiza la misma acción que la anterior cabecera, pero para navegadores más antiguos que no soporten CSP.

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

CABECERA X-POWERED-BY ELIMINADA

El servidor incluía la cabecera "X-Powered-By: PHP/7.2.2", revelando información sobre la versión de PHP utilizada. Esto supone un riesgo de seguridad, ya que un atacante podría aprovechar vulnerabilidades conocidas de esa versión.

Para solucionarlo, se añadió en el **Dockerfile** la siguiente línea para desactivar dicha cabecera en todas las respuestas:

```
RUN echo 'expose_php = Off' > /usr/local/etc/php/conf.d/security.ini
```

Tras reconstruir el contenedor con "docker compose up --build", se comprobó con "curl -I http://localhost:81/ | grep -i powered" que la cabecera ya no se muestra. El nuevo análisis con OWASP ZAP 2.16.1 confirmó que la alerta 10 desaparece.

CABECERA SERVER MODIFICADA

El servidor incluía la cabecera "Server: Apache/2.4.25 (Debian)", que revelaba información sobre la versión exacta del software utilizado. Esta información podía ser aprovechada por atacantes para identificar vulnerabilidades conocidas de esa versión.

Para solucionarlo, se modificó la configuración de Apache en el archivo **Dockerfile**, de manera que solo muestre el nombre del servidor, ocultando los datos de versión. Para ello, se añadió el siguiente código:

```
#Ocultar versión y sistema operativo

RUN echo 'ServerTokens Prod' >
/etc/apache2/conf-available/security.conf && \

   #Evita la firma de servidor en páginas de error

   echo 'ServerSignature Off' >>
/etc/apache2/conf-available/security.conf && \

   #Configuración de cabeceras.

   echo '<IfModule mod_headers.c>' >>
/etc/apache2/conf-available/security.conf && \

   echo '  Header unset Server' >>
/etc/apache2/conf-available/security.conf && \

   echo '  Header always set Server "SecureServer"' >>
/etc/apache2/conf-available/security.conf && \

   echo '</IfModule>' >> /etc/apache2/conf-available/security.conf && \

   a2enconf security
```

Se utiliza el Dockerfile para modificar ésta información en vez de el archivo .htaccess, a pesar de que dicho archivo es, generalmente, el que permite modificar las cabeceras, porque la cabecera de "Server" se escribe antes siquiera de tener en cuenta el archivo .htaccess, a si que hace falta editar un archivo de configuración y moverlo a la carpeta pertinente.

Tras reconstruir el contenedor con "docker compose up --build", se verificó el cambio ejecutando "curl -I http://localhost:81/ | grep -i server". La respuesta muestra únicamente Server: Apache, confirmando que ya no se revela la versión. El nuevo análisis con OWASP ZAP 2.16.1 confirmó que la alerta sobre este problema ha desaparecido.

MANIPULACIÓN DE PARÁMETROS SOLUCIONADA

La alerta sobre manipulación de parámetros ocurre porque en ningún momento se validan los datos del usuario en su registro. Para arreglar esto se han llevado a cabo los siguientes pasos:

1- Comprobar que la acción correspondiente a mandar un formulario es POST. Necesario para comprobar que el usuario manda un formulario legítimo:

```
if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
        http_response_code(405);
        die('método incorrecto.');
    }
```

2- Comprobar que los parámetros introducidos son los esperados. Así se evita que se puedan manipular los campos enviados al servidor:

```
$param_esperados =
['register_submit','csrf_token','nombre','apellido','numDni','letraDni'
,'tlfn','fNacimiento','email','usuario','contrasena'];
    foreach ($_POST as $param => $valor) {
        if (!in_array($param, $param_esperados, true)) {
            http_response_code(400);
            die('parámetro no esperado');
        }
    }
```

3- Sanitizar los datos enviados por formulario. Es decir, limpiar los valores eliminando espacios en blanco o caracteres potencialmente peligrosos:

```
$nombre= htmlspecialchars(trim($_POST['nombre']));
    $apellido = htmlspecialchars(trim($_POST['apellido']));
    ...
```

CABECERA X-CONTENT-TYPE-OPTIONS AÑADIDA

Se ha añadido la cabecera “X-Content-Type-Options” para evitar que el navegador detecte un archivo de forma errónea. Para ello se ha establecido esta cabecera como “nosniff”, de esta forma, el navegador debe interceptar los recursos y comprobar que son legítimos, haciendo que los recursos solo se procesen con el Content Type declarado.

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

DIVULGACIÓN DE ERROR DE APLICACIÓN SOLUCIONADA

Esta alerta ocurre sobre todo en register.php. Si sucede algún error con la base de datos, se muestra información sensible de la aplicación como puede ser el error de conexión, revelando la ubicación de archivos relevantes. Para corregir esto, ha bastado con cambiar las líneas de error de conexión que se tenían por:

```
...
//comprobar la conexión
if ($conn->connect_error) {
    error_log("Connection failed: " . $conn->connect_error); //si error
-> mnsj por pantalla
   die("Connection failed: ");
}
```

```
...
       if ($stmt->execute()) {...}
       else {
           //la instrucción no es válida
           error_log("DB query failed: " . $conn->error);
           echo "Ha ocurrido un error al procesar el registro.";

           $stmt->close();
```

Es decir, lo que se hace ahora es no mostrar la información del error mediante “die(‘Connection failed: ‘);”

COOKIE SIN ATRIBUTOS SAMESITE y HTTPONLY AÑADIDAS

En este caso se trata con alertas parecidas, ya que el problema es la falta de atributos de las cookies de la web. Su solución consiste en habilitar el flag de HttpOnly y poner el valor ‘Strict’ en Samesite. Para ello, se ha creado un nuevo archivo llamado session_config.php que contiene un método para completar los parámetros que necesitamos para iniciar la sesión correctamente donde sea necesario. En nuestro caso, en index.php, login.php y register.php. A continuación se muestra parte del código utilizado para solucionar la alerta:

```
$path = '/; SameSite=Strict';
...
$httponly = true; // Evita acceso desde JavaScript
...
session_set_cookie_params($lifetime,     $path,     $domain,     $secure,
$httponly);
session_start();
```

OTROS CAMBIOS

Se han aplicado las posibles mejoras descritas en la anterior entrega:

- **Contraseñas**. Al registrar el usuario, se guarda un hash de su contraseña, en vez de guardar directamente la contraseña entera. Además, se verifica en login con el método password_verify. También se oculta la contraseña al escribirla cuando el usuario se registra e inicia sesión.
- **Usuario administrador**. Hemos añadido el atributo ‘rol’ a la tabla usuarios. Este atributo puede tener valor ‘admin’ o ‘user’. Por defecto se asigna ‘user’ ya que el administrador no se puede crear desde la aplicación web.
- Solo el usuario administrador puede añadir y eliminar películas.
- Solo usuarios que hayan iniciado sesión pueden modificar películas, ya sean usuarios normales o administradores.
- Los errores al modificar una película ya no causan tener que volver a introducir todos los valores.