

SISTEMAS DE GESTIÓN DE SEGURIDAD DE SISTEMAS DE INFORMACIÓN

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

Euskoróscopo: auditoría de seguridad web

Entrega 2

Adair Gondan Alonso
Sergio Lusa Coria
Javier Arambarri Calvo

Bilbao, 14 de octubre de 2023

Notas

Esta segunda entrega recoge la auditoría realizada al sistema web de **Euskoróscopo** en el marco de la seguridad web. Este trabajo corresponde a la asignatura de Sistemas de Gestión de Seguridad de Sistemas de Información, cursada en el tercer año del grado en Ingeniería Informática de Gestión y Sistemas de Información.

El objetivo de esta segunda parte es buscar y solucionar las vulnerabilidades del sistema web para garantizar los principios de seguridad que debe cumplir.

Se ha utilizado el programa **ZAP** para analizar el sistema en busca de vulnerabilidades.

El sistema puede encontrarse en la rama *entrega_2* del siguiente repositorio:

https://github.com/Adair-GA/Seguridad_Web

Se han seguido los enlaces bibliográficos proporcionados por ZAP para solucionar cada una de las vulnerabilidades.

Índice general

1. Cómo se han realizado las auditorías	4
2. Análisis sobre los resultados ZAP	8
2.1. Absence of Anti-CSRF Tokens	9
2.2. Content Security Policy (CSP) Header Not Set	9
2.3. Missing Anti-clickjacking Header	10
2.4. Cookie No HttpOnly Flag	10
2.5. Cookie without SameSite Attribute	10
2.6. Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	11
2.7. Server Leaks Version Information via "Server" HTTP Response Header	11
2.8. X-Content-Type-Options Header Missing	11
2.9. Authentication Request Identified	12
2.10. Information Disclosure - Sensitive Information in URL	12
2.11. Information Disclosure - Suspicious Comments	12
2.12. Loosely Scoped Cookie	12
2.13. Session Management Response Identified	13
2.14. User Controllable HTML Element Attribute (Potential XSS)	13
3. Análisis sobre la implementación inicial del sistema	14
3.1. Contraseñas inseguras	14
3.2. Almacenamiento de contraseñas	14
3.3. Almacenamiento de datos personales	15
3.4. Consultas SQL	15
3.5. Registro de intentos de inicio de sesión	16
3.6. Aspectos seguros ya desarrollados	16
4. Cambios realizados	17
4.1. Denegación de acceso	17
4.2. Encryption at rest	17
4.3. Almacenamiento de contraseñas mediante "hash + salt"	19
4.4. Política de tamaño y complejidad de contraseñas	21
4.5. reCAPTCHA	21
4.6. Tokens ANTI-CSRF	22
4.7. Adición de cabeceras CSP (Content Security Policy)	23
4.8. Adición de cabecera Anti-clickjacking	24
4.9. Consultas parametrizadas para evitar SQL injection	25
4.10. Configuración HttpOnly en cookies	26
4.11. Configuración SameSite en cookies	26
4.12. Server Leaks Information via "X-Powered-By" HTTP Response Header	26
4.13. Server Leaks Version Information via "Server" HTTP Response Header	27
4.14. X-Content-Type-Options Header Missing	27

4.15. Actualización de la versión PHP	27
4.16. Registro (<i>logs</i>) de inicio de sesión	27
4.17. Dockerfile	28
5. Resumen de seguridad implementada en el sistema	29
6. Análisis sobre las implementaciones de seguridad	31
7. Conclusiones	33
8. Bibliografía	34

1. Cómo se han realizado las auditorías

Las auditorías se han realizado de dos maneras diferentes.

La primera de ella ha sido utilizando la herramienta **ZAP** (<https://www.zaproxy.org/>) para la obtención de vulnerabilidades. Se ha desplegado el proyecto en localhost mediante Docker, y se ha introducido la dirección de localhost en ZAP. Como resultado se han obtenido respuestas que se han analizado minuciosamente con el objetivo de solucionar esos problemas de seguridad. Asimismo también se ha leído sobre cómo consigue este programa detectar esas vulnerabilidades, puesto que tras implementar correctamente algunas soluciones hemos detectado que las seguía alertando.

Al entrar en ZAP, decidimos no guardar la sesión:

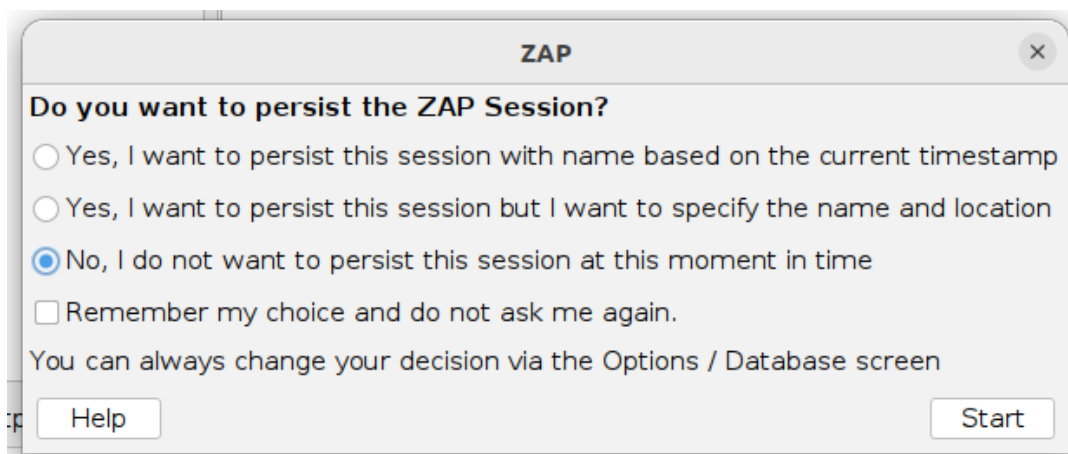


Figura 1.1: *Sesión ZAP.*

Seleccionamos “Automated Scan”:

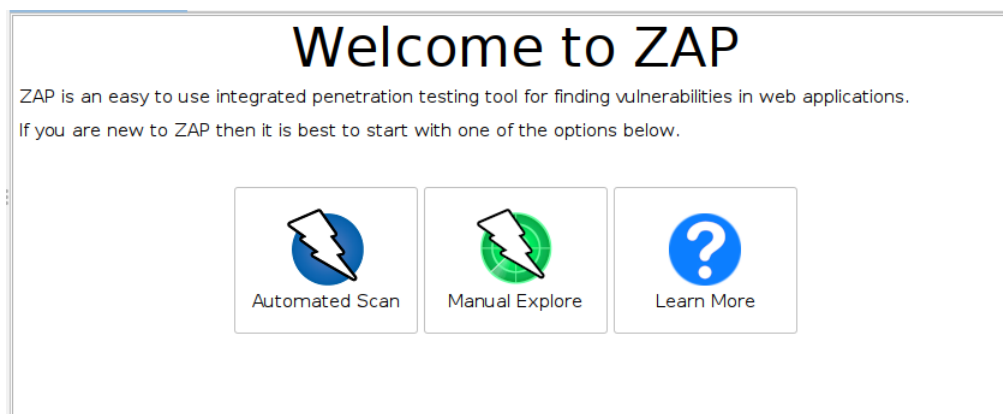


Figura 1.2: *Automated Scan.*

Con el proyecto Docker desplegado, ingresamos la URL del proyecto: `http://localhost:81/`. Hacemos click en “Attack”.

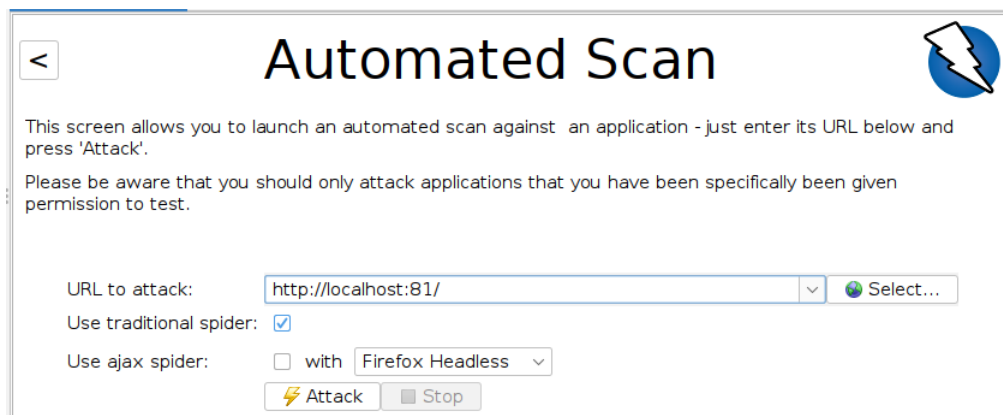


Figura 1.3: *Introducir URL en ZAP.*

Abajo a la izquierda seleccionamos la ventana “Alerts” y nos aparecerán las alertas de seguridad identificadas por ZAP. Estas se catalogan en cuatro grupos: alta prioridad (roja), media prioridad (naranja), baja prioridad (amarilla) e informativa (azul).

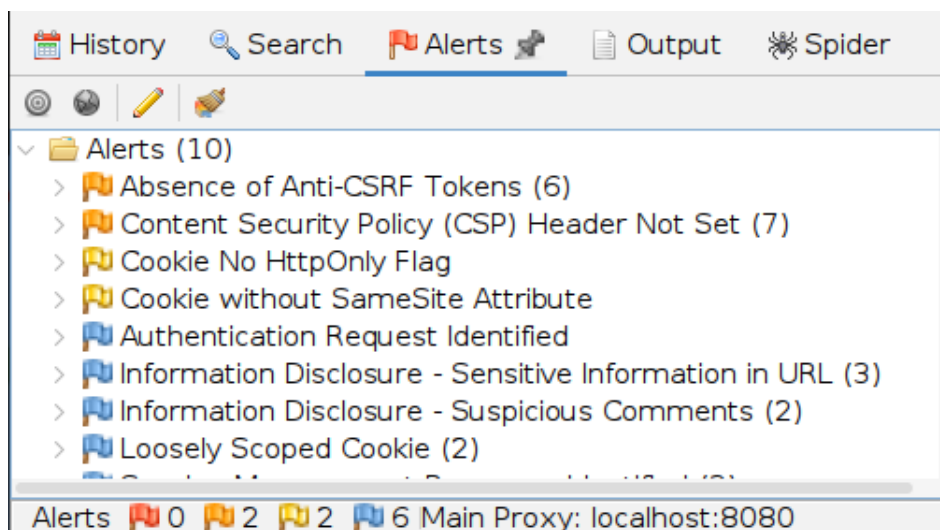


Figura 1.4: *Alertas en ZAP.*

Haciendo click en cada una de ellas se mostrarán sus detalles e información relativa que ayuda a comprender el origen de la alerta, soluciones posibles y referencias a enlaces bibliográficos.

The image shows a screenshot of the 'Edit Alert' dialog box in ZAP. The dialog has a title bar with 'Edit Alert' and a close button. The main content area contains several fields and sections:

- Alert Name:** A dropdown menu showing 'Absence of Anti-CSRF Tokens'.
- URL:** A text field containing 'http://localhost:81/createEntry.php'.
- Risk:** A dropdown menu showing 'Medium'.
- Confidence:** A dropdown menu showing 'Low'.
- Parameter:** A dropdown menu.
- Attack:** A text field.
- Evidence:** A text field containing '<form name="entry" id="entryForm">'.
- CWE ID:** A dropdown menu showing '352'.
- WASC ID:** A dropdown menu showing '9'.
- Description:** A text area containing:

No Anti-CSRF tokens were found in a HTML submission form.
A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target
- Other Info:** A text area containing:

No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf token, csrf, csrfSecret, csrf magic, CSRF.
- Solution:** A text area containing:

Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
- Reference:** A text area containing:

<http://projects.webappsec.org/Cross-Site-Request-Forgery>
<https://cwe.mitre.org/data/definitions/352.html>

At the bottom right, there are two buttons: 'Cancel' and 'Save'.

Figura 1.5: *Detalles de alerta en ZAP.*

La segunda manera ha sido utilizar el conocimiento de la implementación del sistema para identificar los aspectos vulnerables. Además, cabe destacar que en la primera fase del proyecto se implementó este mismo sistema web pero sin reparar a la seguridad, esto es, lo más inseguro posible. Este es el motivo por el que se conocían diversos fallos de seguridad y que deberían ser subsanados en esta segunda fase.

Se han utilizaod los conceptos de seguridad web estudiados a los largo de la asignatura.

Por último, una vez las principales brechas de seguridad estaban identificadas e implementadas se ha vuelto a hacer un escáner con ZAP para comprobar y demostrar la corrección de las implementaciones de seguridad realizadas.

2. Análisis sobre los resultados ZAP

Se han realizado varios análisis ante la posibilidad de obtener diferentes resultados en cada una de ellas, aunque no ha sucedido, por lo que a continuación se exponen las alertas mostradas por ZAP en uno de esos análisis.

Se han detectado un total de catorce alertas: tres de *media prioridad*, cinco de baja prioridad, seis de prioridad informativa y ninguna de **prioridad alta**:

- *Absence of Anti-CSRF Tokens (6)*
- *Content Security Policy (CSP) Header Not Set*
- *Missing Anti-clickjacking Header*
- Cookie No HttpOnly Flag
- Cookie without SameSite Attribute
- Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)
- Server Leaks Version Information via "Server" HTTP Response Header Field
- X-Content-Type-Options Header Missing
- Authentication Request Identified
- Information Disclosure - Sensitive Information in URL
- Information Disclosure - Suspicious Comments
- Loosely Scoped Cookie
- Session Management Response Identified
- User Controllable HTML Element Attribute (Potential XSS)

2.1. Absence of Anti-CSRF Tokens

¿Qué son los Tokens Anti-CSRF?

Son cadenas de caracteres aleatorios muy extensas que permiten evitar los ataques CSRF (*Cross Site Request Forgery*, falsificación de petición en sitios cruzados). Es decir, es un mecanismo que impide que las peticiones maliciosas o no autorizadas cumplan su objetivo. Puede verse un ejemplo muy ilustrativo en la siguiente web: <https://backtrackacademy.com/articulo/anti-csrf-tokens-para-prevenir-cross-site-request-forgery-csrf>.

Por tanto, los formularios de envío de las páginas *createEntry.php* y *login.php* son vulnerables a este tipo de ataques, entre otras peticiones. Al hacer click en “Eliminar” para borrar una entrada también es susceptible a este tipo de ataques.

Por ejemplo, un ataque CSRF en nuestro sistema podría consistir en la creación de una entrada desde un usuario que no ha introducido esa entrada. En el sistema se reflejaría que se ha realizado esa petición, cuando en realidad ha sido maliciosa. Si se dispone de Tokens, la petición maliciosa será rechazada.

2.2. Content Security Policy (CSP) Header Not Set

¿Qué es la cabecera de Política de Seguridad del Contenido?

Es una cabecera de respuesta del protocolo HTTP con el fin de aumentar la seguridad del sistema web. Permite restringir qué recursos y de qué orígenes pueden cargarse. Puede implementarse a través de la etiqueta **meta** de HTML.

La Política de Seguridad del Contenido fue principalmente concebida para impedir, evitar o minimizar los ataques *Cross-Site scripting (XSS)*, y posteriormente, ataques *clickjacking*.

Los ataques *XSS* consisten en implantar scripts maliciosos en páginas web seguras y de confianza. Esos scripts son ejecutados en el navegador web del usuario.

Los ataques *clickjacking* consisten en introducir una capa invisible entre la capa original y la interacción del usuario. Es decir, mientras el usuario cree estar en el sitio oficial, realmente está haciendo click o introduciendo datos en una capa invisible, aunque visualice la capa original.

Por tanto, el sistema web deberá incluir la cabecera *CSP* en todos aquellos ficheros .html o .php.

2.3. Missing Anti-clickjacking Header

Una cabecera *Anti-clickjacking* es un elemento que impide los ataques de tipo *click-jacking*, explicados en el apartado anterior. De hecho, esta cabecera forma parte de la Política de Seguridad del Contenido. Sin embargo, esta no puede ser implementada a través de la etiqueta HTML *meta*, por lo que se evalúa por separado.

En este caso, se debe modificar la configuración de Apache para activar este tipo de cabeceras.

2.4. Cookie No HttpOnly Flag

HttpOnly es un *flag* adicional que se incluye en una cabecera de respuesta de HTTP *Set-Cookie*. Esta cabecera se utiliza cuando se genera una *cookie* y ayuda a mitigar el riesgo de que el script del lado del cliente acceda a la cookie protegida. Esto solo sucede si el navegador lo soporta, mas prácticamente la totalidad de ellos lo hace. En otras palabras, si se incluye el flag HttpOnly en la cabecera de respuesta HTTP, que es opcional, no se puede acceder a la cookie a través del script del lado del cliente. Además, incluso si existiese un fallo de *Cross Site Scripting (XSS)*, y un usuario accede accidentalmente a un enlace que explota este fallo, el navegador no revelará la cookie a terceros.

Si un navegador no admite HttpOnly y un sitio web intenta establecer una cookie *HttpOnly*, el navegador ignorará el indicador HttpOnly, creando así una *cookie* tradicional accesible mediante script. Como resultado, la *cookie* (normalmente su *cookie* de sesión) se vuelve vulnerable al robo o modificación por parte de un script malicioso.

En resumen, la ausencia del *flag* HttpOnly permite acceder a la *cookie* correspondiente desde un *script* del lado del cliente. En caso de producirse un ataque *Cross Site Scripting (XSS)*, los delincuentes tendrían acceso al contenido de las *cookies*.

Por ello, este *flag* es parte de la protección contra ataques *XSS*.

2.5. Cookie without SameSite Attribute

SameSite es un atributo que se incluye en una cabecera de respuesta de HTTP *Set-Cookie*. Este atributo permite restringir la existencia, envío y acceso a la *cookie* a un contexto determinado. El contexto o sitio puede ser de dos tipos: *Strict* o *Lax*.

Strict proporciona que la *cookie* solamente se envíe en un contexto propio, es decir, si el sitio de la *cookie* coincide con la URL que se muestra en el navegador. Esto implica que al acceder al sitio mediante un enlace externo, por ejemplo, desde un correo electrónico, no se enviará la *cookie*.

Lax, por su parte, como su propio nombre indica, es una política más laxa y permite

enviar la *cookie* al acceder de forma externa. Es la configuración adecuada para visualizar el sitio de forma externa, por ejemplo, indexado en un correo electrónico.

2.6. Server Leaks Information via “X-Powered-By” HTTP Response Header Field(s)

X-Powered-By es una cabecera de respuesta HTTP que muestra información sobre la tecnología que se usa en nuestro sistema. Es por ello por lo que los atacantes podrían intentar explotar vulnerabilidades de esa tecnología para atacar indirectamente nuestro sistema, puesto que las vulnerabilidades que afecten a esa tecnología repercutirán en nuestro sistema.

Se trata de una configuración propia del servidor, por lo que será este quien la active, desactive o manipule. Para no mostrar información relevante, el servidor no debe enviar esta cabecera en la respuesta HTTP.

2.7. Server Leaks Version Information via “Server” HTTP Response Header

Al igual que la cabecera anterior, desvela información sensible, en este caso, sobre la versión del servidor que se usa. Consecuentemente, los delincuentes podrían atacar las vulnerabilidades presentes en dicha versión del servidor para atacar nuestro sistema. Para evitarlo, hay que eliminar la cabecera *Server* de la respuesta HTTP del servidor.

2.8. X-Content-Type-Options Header Missing

X-Content-Type-Options es una cabecera de respuesta HTTP que permite evitar que los navegadores adivinen los tipos de medios “MIME”. De esta forma, posibilita impedir ataques de tipo *content type sniffing*.

En este tipo de ataques, los atacantes engañan a los navegadores web para que interpreten los archivos de una forma distinta a su intención original, lo que provoca problemas de seguridad como los ataques *XSS*, entre otros.

2.9. Authentication Request Identified

Es una alerta de tipo informativo que indica que una determinada solicitud se ha identificado como una solicitud de autenticación. Esto lo detecta mediante la observación de los campos como “usuario” y “contraseña”.

No requiere ninguna modificación ni representa ninguna vulnerabilidad.

2.10. Information Disclosure - Sensitive Information in URL

Es una alerta de tipo informativo. Indica que una solicitud contiene información potencialmente confidencial en la URL. En nuestro caso, nos lo marca con el parámetro *token* de los formularios, por lo que corresponde con el valor de los *token anti-csrf*.

No obstante, la comprobación de estos *token* se realiza comparando su hash, es decir, comprobando la integridad. Consecuentemente, no es necesario modificar nada debido a que no supone un riesgo.

2.11. Information Disclosure - Suspicious Comments

Es una alerta de tipo informativo con el objetivo de prevenir insertar comentarios en el código que filtren información o datos sensibles acerca del funcionamiento del sistema. La herramienta ZAP nos alerta en el fichero *createEntry.js*, concretamente en el comentario:

“//Si el formato de la fecha es correcto, proceder a crear la entrada en la DB”.

Ha relacionado la palabra “DB” con “Database”, lo que ha ocasionado la alerta. Tras analizarlo, concluimos que no se pone en peligro la seguridad del sistema.

2.12. Loosely Scoped Cookie

Es una alerta de tipo informativo que nos indica que las *cookies* del sistema han de ser establecidas a un alcance completamente cualificado o de confianza.

Las *cookies* de un sistema pueden tener un ámbito de dominio o de ruta, es decir, dicho ámbito determina qué dominios pueden acceder a la misma. Por tanto, debemos asegurarnos de que el ámbito de las *cookies* es seguro, y corresponde a dominios certificados.

2.13. Session Management Response Identified

Es una alerta de tipo informativo que nos indica que la respuesta a las peticiones contiene *tokens* o información acerca de una sesión. Identifica la *cookie* de sesión en las respuestas de las cabeceras.

No hay nada que solucionar.

2.14. User Controllable HTML Element Attribute (Potential XSS)

Es una alerta de tipo informativo que nos alerta de posibles valores de atributos controlables HTML, lo que podría conllevar una vulnerabilidad ante ataques *XSS*.

La solución consiste en validar todas las entradas y sanitizar la salida antes de escribir en cualquier atributo HTML.

3. Análisis sobre la implementación inicial del sistema

Además de los resultados obtenidos de la herramienta ZAP, existen otras vulnerabilidades conocidas debido a cómo se desarrolló el código en la primera entrega del trabajo. Esas vulnerabilidades se recogen en este apartado.

3.1. Contraseñas inseguras

No existe ninguna política de tamaño ni de complejidad de contraseñas, es decir, no se solicita que la contraseña de inicio de sesión cumpla con algunos requisitos para fortalecerla.

Es recomendable que las contraseñas estén formadas, al menos, por caracteres en mayúsculas, en minúsculas, números y algún carácter especial.

3.2. Almacenamiento de contraseñas

En el desarrollo inicial del sistema las contraseñas se almacenan sin ninguna protección ni ocultación ante accesos fraudulentos, es decir, se guardan literalmente. Este procedimiento es altamente vulnerable, pues se trata de un fallo criptográfico.

La solución consiste en no almacenar la contraseña literalmente, sino su *hash*. Además, en caso de sufrir un ataque a la base de datos o un robo de información, no se podrá calcular la contraseña partiendo de su resumen *hash*, puesto que estos algoritmos no tienen función inversa.

Adicionalmente, es necesario añadir una sal (*salt*) a la contraseña para evitar que se puedan precalcular los resúmenes *hash* de todo el espacio de contraseñas del sistema. La sal es una cadena de caracteres aleatorios y única para cada usuario que se añade a la contraseña previo a calcular el *hash*. De esta manera, aquellos usuarios con la misma contraseña tendrán un valor diferente almacenada en la base de datos, impidiendo la identificación de aquellos usuarios con las misma clave de acceso.

3.3. Almacenamiento de datos personales

Al igual que las contraseñas, estos se almacenan en formato literal, por lo que están expuestos ante posibles ataques o robos de información.

Asimismo, es imprescindible cumplir con la legislación vigente, en este caso con el Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016, relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y con la adaptación española, la Ley Orgánica 3/2018, conocida como LOPD: Ley Orgánica de Protección de Datos. Concretamente, el considerando 83 del Reglamento hace referencia al cifrado como posible medida de protección y los artículos 6, 32 y 34 profundizan en el tema para cuando los datos recogidos se utilizan para otro fin, con el objetivo de garantizar el anonimato mediante la técnica de la seudonimización.

Este Reglamento Europeo deroga la Directiva 95/46/CE (Reglamento general de protección de datos) y la adaptación española no hace ninguna referencia al cifrado de los datos, por lo que deberemos regirnos por la normativa del Parlamento Europeo.

La LOPD recoge el mecanismo sancionador en caso de cumplir con lo expuesto, así como el deber de comunicar a los afectados la filtración de sus datos personales en caso de producirse.

Para cumplir con la legislación vigente y proteger a nuestros usuarios, deberemos cifrar los datos *at rest*.

3.4. Consultas SQL

Las sentencias SQL que nos permiten obtener los datos solicitados de la base de datos se ejecutan de manera interpretada, debido a que la API utiliza el intérprete. Como se ha estudiado en el tema de Seguridad Web de la asignatura, esto constituye un típico fallo de seguridad denominado “Inyección SQL”.

Mediante este ataque se añaden parámetros o atributos interpretables a la sentencia que generan una respuesta no deseada por el sistema, exponiendo los datos ante intrusos.

La solución es parametrizar las sentencias, esto es, a la hora de ejecutarlas introducir los parámetros correspondientes de manera que no se interpreten, sino que se admitan de manera literal.

3.5. Registro de intentos de inicio de sesión

Los intentos de inicio de sesión no se registran y no se puede llevar un control acerca de los inicios de sesión e intentos.

Cuando estos son incorrectos en numerosas ocasiones en un corto periodo temporal puede deducirse que se debe a un ataque de fuerza bruta o *Credential Stuffing*, entre otros. Sin registrarlos resulta prácticamente imposible (sí podríamos saber el elevado número de peticiones, pero ningún dato relevante) comprobar si los accesos o los intentos han sido legítimos o no, o si estamos sufriendo alguno de esos ataques.

A este proceso se le llama “Registro de *logs*” y en el caso de, inicio de sesión, nos registrará la información importante sobre los intentos.

Los *logs* son aplicables y escalables al control y análisis de cualquier otra actividad sospechosa que podría producirse. También pueden registrarse los errores que se dan, si se dan con un usuario registrado, con quién, la fecha y la hora...

3.6. Aspectos seguros ya desarrollados

Por otra parte, destacar que el sistema ya cumplía con algunos requisitos de seguridad y se presentan a continuación ordenados por el tipo de vulnerabilidad al que pertenecen.

- **Rotura de control de acceso:** no se permite al acceso ni a través de la interfaz del usuario ni mediante un URL a la página “Modificar datos” si no está iniciada la sesión.
- **Componentes vulnerables y obsoletos:** se utiliza la versión PHP 7.2.2, que elimina la extensión MySQL y la sustituye por MySQL Extendida (mysqli) desde la versión 7.0.0.
- **Fallos de identificación y autenticación:** se elimina la sesión después de desconectarse mediante *session_destroy()* y se elimina la *cookie* correspondiente. Además, se sanitizan los valores introducidos en el formulario de manera que si no cumplen con los requisitos (por ejemplo, introducir números en el nombre) no puede lanzarse la petición de registro.
- **Peticiones POST y no GET:** se utiliza el método POST para enviar la información al servidor a través del cuerpo de la petición HTTP, en lugar del URL como hace el método GET.

4. Cambios realizados

Se han aplicado mejoras que solucionan todas las vulnerabilidades expuestas en los apartados uno y dos. En esta sección se exponen, agrupados por grupos generales de vulnerabilidades, todos los cambios implementados.

4.1. Denegación de acceso

Se ha añadido la denegación de acceso mediante URL a la página de Inicio de Sesión y Registro si se tiene una sesión iniciada.

También se registran los intentos de inicio de sesión y no hay archivos de metadatos ni de copias de seguridad en el nivel root del sistema.

4.2. Encryption at rest

Se han cifrado todos aquellos datos susceptibles a ataques por su gran valor, es decir, los datos personales. Los datos de los horóscopos no se han cifrado debido a que no representan ningún dato personal, y en todo caso pueden ser inventados.

Para ello, por simplicidad, utilizamos un cifrado simétrico y todas las operaciones se hacen en el servidor, tanto encriptar como desencriptar. Cuando se van a insertar datos en la base de datos, estos son cifrados; y cuando se solicitan son descifrados.

En este caso y por comodidad, la clave la almacenamos en el propio directorio de la aplicación. En una implementación real se debería buscar otra ubicación, así como dispositivos de hardware dedicado que dejan de funcionar cuando hay un acceso físico. Se ha bloqueado configurando un archivo *.htaccess* el acceso mediante URL tanto a la clave de cifrado y descifrado como a su directorio.

Veamos un ejemplo del desarrollo implementado:

```
function encrypt(string $plaintext, string $filePath){
    $keyPath=$filePath."key.pem";
    $ivPath=$filePath."iv.txt";
    $cipher = "aes-256-cbc";
    $key = file_get_contents($keyPath);
    $iv = file_get_contents($ivPath);
    $iv = base64_decode($iv);
    $output = openssl_encrypt($plaintext, $cipher, $key, 0, $iv);
    $output = base64_encode($output);
    return $output;}
```

```
function decrypt(string $ciphertext, string $filePath){
    $keyPath=$filePath."key.pem";
    $ivPath=$filePath."iv.txt";
    $cipher = "aes-256-cbc";
    $key = file_get_contents($keyPath);
    $iv = file_get_contents($ivPath);
    $iv = base64_decode($iv);

    $output = openssl_decrypt(base64_decode($ciphertext),
                              $cipher, $key, 0, $iv);

    return $output;
}
```

Se emplea el algoritmo de cifrado “AES-256-CBC”, esto es, *Advanced Encryption Standard* de 256 bits y con la operación *Cipher-Block Chaining*. Se trata de un algoritmo de cifrado por bloques. Los bloques son siempre del mismo tamaño. Además, los algoritmos de cifrado por bloques requieren de un vector de inicialización (IV) de 16 bytes.

Pero, ¿qué es un IV?

Es el estado inicial, que se añade al cifrado para ocultar patrones en los datos cifrados, permitiendo evitar la necesidad de volver a emitir una nueva clave después de cada invocación al algoritmo.

En la siguiente figura se muestra cómo quedan almacenados los datos en la base de datos.

dni	nombre
UjNRc01kNGVjTHU0b1NVZ05mdFRDZz09	ampqeStEOTVIOExLQWNZM1VRUHU0UT09
UXZ2V0dPeFwcwzA3NVV4Q0F2OHNsQT09	djE4eDk3eWJ1aXdrei9ZeDg0MnlvQT09

Figura 4.1: *Ejemplo del cifrado de datos personales.*

4.3. Almacenamiento de contraseñas mediante “hash + salt”

Se ha mejorado la seguridad del portal de acceso de la plataforma. La base de datos no almacenará las contraseñas de los usuarios, sino la combinación de su *hash* con el *salt* (valor aleatorio asociado a un usuario).

De esta manera, cuando el usuario introduzca su usuario y contraseña, el sistema juntará la contraseña introducida con el *salt* asociado a ese usuario y calculará el resumen *hash*. Si resulta el mismo que el almacenado, se iniciará la sesión.

No se podrá saber cuál es la contraseña de un usuario, puesto que no se almacenan. En cambio, el *salt* de cada usuario sí se almacena en la base de datos.

El *salt* se genera de manera aleatoria a partir de un *string* que contiene todos los caracteres posibles que pueden formar parte. Este proceso debe realizarse con la mayor entropía posible. En este caso, se ha optado por una cadena sencilla preestablecida de diversos caracteres debido al ámbito formativo del proyecto.

Para obtener los caracteres que formarán parte se obtienen tantos números aleatorios como se desee (longitud del *salt*). Estos números aleatorios representan el índice de cada carácter dentro del *string*, permitiendo construir el *string* del *salt*.

Se concatena la contraseña introducida con el *salt* generado y se calcula el *hash* de la combinación.

Además, esta mejora en la seguridad ha ocasionado tener que suprimir la creación de la sentencia SQL en `js/modifyUserData.js` para realizarla en el servidor, en `api/update_data.php`.

Los archivos modificados han sido los siguientes:

- `api/login.php`
- `api/signup_register.php`
- `js/modifyUserData.js`
- `api/update_data.php`

Porción de código para el inicio de sesión (PHP):

```
$query = "SELECT sal FROM usuarios WHERE (usuario = '$usuario' OR email  
      = '$usuario')";  
$result = mysqli_query($conn, $query) or die (mysqli_error($conn));  
$row = mysqli_fetch_array($result);  
$contrasena .= $row[0]; // Concatenamos la contraseña introducida con la sal  
$contrasena = hash('sha512', $contrasena); // Obtenemos el hash  
      correspondiente  
$query = "SELECT email, usuario, dni FROM usuarios WHERE (usuario =  
      '$usuario' OR email = '$usuario') AND contraseña = '$contrasena'";
```

Porción de código para el registro (PHP):

```
function getSalt($n) {
    $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
    UVWXYZ';
    $randomString = '';

    for ($i = 0; $i < $n; $i++) {
        $index = rand(0, strlen($characters) - 1);
        $randomString .= $characters[$index];
    }

    return $randomString;
}
...

$salt = getSalt(6);
$contrasena .= $salt;
$contrasena = hash('sha512', $contrasena);

...

$query = "INSERT INTO usuarios (dni, nombre, apellidos, usuario, contraseña,
    sal, email, telefono, fecha_nacimiento)
    VALUES ('$dni', '$nombre', '$apellido', '$usuario',
    '$contrasena', '$salt', '$email', '$telefono',
    '$fecha_nacimiento')";
```

Porción de código para modificar la contraseña (PHP):

```
$query = "UPDATE `usuarios` SET";
function getSalt() {
    include '../dbconn.php';
    $userForSalt = $_SESSION['usuario'];
    $query = "SELECT sal FROM usuarios WHERE usuario = '$userForSalt'";
    $result = mysqli_query($conn, $query) or die (mysqli_error($conn));
    $row = mysqli_fetch_array($result);

    return $row[0];
}
...
```

```

if ($contrasena!=""){
    $salt = getSalt();
    $contrasena .= $salt;
    $contrasena = hash('sha512', $contrasena);

    $add=" contraseña = '" . $contrasena;
    $query.=$add;
    $query.="',";
}

...
$result = mysqli_query($conn, $query) or die("Error in Selecting " .
mysqli_error($conn));

```

4.4. Política de tamaño y complejidad de contraseñas

Se ha introducido la siguiente política para las contraseñas. Deberán contener, al menos:

- Una letra minúscula.
- Una letra mayúscula.
- Un número.
- Un carácter especial (@!%*?&).
- Una longitud mínima de seis caracteres.

4.5. reCAPTCHA

Se ha incorporado en los formularios un *reCAPTCHA* como medida de seguridad para evitar peticiones realizadas por bots y que estas no colapsen el sistema.

reCAPTCHA es un sistema de verificación de la empresa Google que solicita superar una determinada prueba antes de enviar el formulario. Funciona mediante una clave pública y otra privada. La clave pública se expone en el lado del cliente, en el formulario; mientras que la privada se almacena en el servidor para comprobar que la prueba se ha superado exitosamente.

4.6. Tokens ANTI-CSRF

Se han implementado tokens anti-CSRF para evitar este tipo de ataques en nuestro sistema web. Para ello, los formularios disponen de un *input* oculto llamado “token”, cuyo valor es una cadena de caracteres aleatorios de veinticuatro (24) dígitos, por ejemplo: `959c6ddef66a7e43c012d62d0ed5f4061054bc444e14c978`.

El valor del “token” se genera iniciando una sesión PHP al cargar cada una de las páginas, al inicio de estas, y se almacena en la variable `$_SESSION`. Cuando se hace la llamada a la API correspondiente, esta compara el “token” de la variable `$_SESSION` con el valor que ha recibido a través de la `$_REQUEST` (valor del *input* oculto de los formularios, es decir, token de cada uno de los formularios). La comparación se realiza mediante el *hash* de ambos valores, evitando la interceptación del valor de comparación.

Los archivos modificados han sido los siguientes:

- `index.php`
- `createEntry.php`
- `login.php`
- `modifyEntry.php`
- `modifyUserData.php`
- `js/index.js`
- `js/login.js`
- `js/modifyEntry.js`
- `js/modifyUserData.js`
- `api/create_entry.php`
- `api/login.php`
- `api/signup_register.php`
- `api/update_data.php`
- `api/update_entry.php`

Al inicio de cada página PHP se genera el “token”:

```
<?php
session_start();
$_SESSION['token'] = bin2hex(random_bytes(24));
?>
```

Salvo en aquellas en las que se requiere el inicio de sesión, puesto que al iniciar sesión se genera un “token” que corresponde a ese usuario, y se almacena también en la variable `$_SESSION`:

```

...
session_start();
$_SESSION['email'] = $row[0];
$_SESSION['usuario'] = $row[1];
$_SESSION['dni'] = $row[2];
$_SESSION['token'] = bin2hex(random_bytes(24));
echo "Login correcto";
...

```

Cuando una API recibe alguna petición, antes de ejecutarla comprueba el “token”, evitando ataques CSRF:

```

$token = $_REQUEST['token'];
if (hash_equals($token, $_SESSION['token'])) {
...
} else {
header($_SERVER['SERVER_PROTOCOL'] . ' 405 Method Not Allowed');
exit;
}

```

4.7. Adición de cabeceras CSP (Content Security Policy)

Se ha añadido la cabecera

```

<meta http-equiv="Content-Security-Policy" content="default-src 'self'
cdn.jsdelivr.net">

```

a los archivos `createEntry.php`, `index.php`, `login.php`, `modifyEntry.php` y `modifyUserData.php`.

Adicionalmente, debido a la implementación del Captcha se han incluido los dominios necesarios a la cabecera de `index.php`:

```

www.google.com www.gstatic.com

```

Nuestro sistema web sólo permitirá contenido propio (*'self'*), de *cdn.jsdelivr.net*, de *www.google.com* y de *www.gstatic.com* por el Captcha. No permitirá scripts ni estilos que esté entre líneas.

Por eso mismo se han sustituido los *onclick* de los botones por *eventListeners*, de esta manera se ha suprimido el atributo inseguro *'unsafe-inline'* de los ficheros.

A continuación se explican cada uno de los atributos que conforman la cabecera:

- **default-src:**

indica que se utilizará la configuración que contenga para aplicarlo al resto del contenido de la CSP, puesto que se omiten (*style-src*, *image-src*...).

- **'self':**
permite coger contenido propio del sistema web.
- **'unsafe-inline':**
permite interpretar los estilos y scripts entre líneas, indexados dentro de las etiquetas HTML. Debe ser sustituido por el uso de 'nonce' o 'hash' o incluir esas partes en archivos separados, como se ha hecho.
- **cdn.jsdelivr.net:**
permite indexar contenido proveniente de ese dominio, usado para la plantilla de Bootstrap.
- **www.google.com www.gstatic.com:**
permite indexar contenido de esas fuentes, en nuestro caso, el Captcha.

4.8. Adición de cabecera Anti-clickjacking

Para solucionar esta vulnerabilidad, debemos añadir

`frame-ancestors 'none'`

en la CSP, pero no puede hacerse directamente en la cabecera de

`<meta>`

porque esta no lo soporta. Por lo que

`<meta http-equiv="Content-Security-Policy" content="frame-ancestors 'none'">`

no nos sirve. Tendremos que hacer que el servidor devuelva la cabecera para todas las páginas. En nuestro caso, usamos Apache, por lo que deberemos modificar el fichero de configuración `/etc/apache2/conf-enabled/security.conf` y descomentar la línea 70:

`Header set X-Frame-Options: "sameorigin"`

A continuación activaremos el módulo de "headers" en Apache y reiniciamos para activar la nueva configuración:

```
$ sudo a2enmod headers
$ sudo systemctl restart apache2
```

Pero si queremos aplicar estos cambios al docker, deberemos modificar el dockerfile y crear un archivo de configuración que se copiará en el contenedor, quedando de la siguiente manera el dockerfile:

```
FROM php:8.2.12-apache
RUN docker-php-ext-install mysqli
RUN a2enmod headers
RUN service apache2 restart
COPY security.conf /etc/apache2/conf-enabled
```

4.9. Consultas parametrizadas para evitar SQL injection

Se han parametrizado las consultas de los siguientes archivos: `api/create_entry.php`, `api/login.php`, `api/signup_register`, `api/update_data.php` y `modifyEntry.php`.

Por ejemplo, en `api/create_entry.php` se ha realizado la siguiente modificación:

```
//Sin PARAMETRIZAR
$query = "INSERT INTO horoscopos (nombre, fecha_nacimiento, signo_solar,
    signo_lunar, mercurio_retrogrado) VALUES ('$nombre',
    '$fecha_nacimiento', '$signosolar', '$signolunar',
    '$boolRetrogrado')";
```

```
$result = mysqli_query($conn, $query) or die (mysqli_error($conn));
```

```
//CON PARAMETRIZACIÓN
```

```
$query = "INSERT INTO horoscopos (nombre, fecha_nacimiento, signo_solar,
    signo_lunar, mercurio_retrogrado) VALUES (?, ?, ?, ?, ?)";
```

```
$stmt = mysqli_prepare($conn, $query) or die (mysqli_error($conn));
mysqli_stmt_bind_param($stmt, "ssssi", $nombre, $fecha_nacimiento,
    $signosolar, $signolunar, $boolRetrogrado);
mysqli_stmt_execute($stmt);
```

```
$result = mysqli_stmt_get_result($stmt);
```

Además, se ha eliminado la construcción de las sentencias SQL en el lado del cliente (en los scripts de JavaScript) para hacerlo en el lado del servidor, concretamente en sus respectivas APIs.

4.10. Configuración HttpOnly en cookies

Dado que sólo se utiliza una cookie en el sistema web (la variable “\$_SESSION” en PHP), sólo se ha tenido que incluir la siguiente configuración en la cabecera HTTP, concretamente en index.php, que es la parte del sistema a la que se accede por primera vez.

```
<?php
    ini_set("session.cookie_httponly", True);
    session_start();
    ... ?>
```

Con el comando `ini_set` estamos indicando que la cookie `$_SESSION` tendrá la configuración de `HttpOnly` activada (*true*).

4.11. Configuración SameSite en cookies

Para versiones superiores a PHP 7.3, se puede hacer de la misma forma que `HttpOnly`. Si no, habrá que modificar la configuración de Apache.

En nuestro caso, hemos actualizado PHP a 8.2.12, y añadido la siguiente línea de código en index.php.

```
ini_set("session.cookie_samesite", "Strict");
```

4.12. Server Leaks Information via "X-Powered-By" HTTP Response Header

Se incluye la siguiente sentencia en el fichero `security.conf` que después mediante el `dockerfile` se copia en el contenedor y se aplica al servidor Apache del contenedor.

```
Header unset X-Powered-By
```

4.13. Server Leaks Version Information via "Server" HTTP Response Header

Se incluyen la siguientes sentencias en el fichero `apache2.conf` que después mediante el `dockerfile` se copia en el contenedor y se aplica al `apache` del contenedor.

```
ServerSignature Off
ServerTokens Prod
```

Asimismo, se incluye la siguiente sentencia en `security.conf` para desactivar y asegurarnos de ello (es opcional):

```
Header unset Server
```

4.14. X-Content-Type-Options Header Missing

Se incluye la siguiente sentencia en el fichero `security.conf` que después mediante el `dockerfile` se copia en el contenedor y se aplica al `apache` del contenedor.

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

4.15. Actualización de la versión PHP

Se ha actualizado a la versión 8.2.12 de PHP para poder soportar las cabeceras de seguridad introducidas en los pasos anteriores.

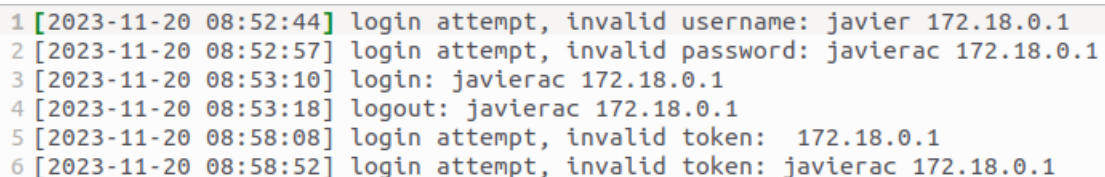
4.16. Registro (*logs*) de inicio de sesión

Se registran los intentos de inicio de sesión en un *log*, tanto los exitosos como los fallidos junto con su error. El error podrá consistir en el *token* anti-CSRF o credenciales incorrectas. También se almacena la fecha y hora del intento, el usuario con el que se ha intentado acceder y la dirección IP del origen.

```
$log = fopen("../logs/log.txt", "a");
$today = date("Y-m-d H:i:s");
fwrite($log, "[ $today ] $mss: " . $user . " " . $_SERVER['REMOTE_ADDR'] . "\n");
fclose($log);
echo $echo;
```

En función del error, el mensaje será diferente:

```
$mss = "login";  
$mss = "logout";  
$mss = "login attempt";  
$mss = "login attempt, invalid password";  
$mss = "login attempt, invalid username";  
$mss = "login attempt, invalid token";
```



```
1 [2023-11-20 08:52:44] login attempt, invalid username: javier 172.18.0.1  
2 [2023-11-20 08:52:57] login attempt, invalid password: javierac 172.18.0.1  
3 [2023-11-20 08:53:10] login: javierac 172.18.0.1  
4 [2023-11-20 08:53:18] logout: javierac 172.18.0.1  
5 [2023-11-20 08:58:08] login attempt, invalid token: 172.18.0.1  
6 [2023-11-20 08:58:52] login attempt, invalid token: javierac 172.18.0.1
```

Figura 4.2: *Logs.*

4.17. Dockerfile

Tanto para añadir en Apache las cabeceras descritas, como para dar a Apache permisos de escritura y poder registrar los inicios de sesión, ha sido necesario modificar el dockerfile.

Se activa el módulo de cabeceras, se dan permisos de escritura en la carpeta “www-data” y la configuración se detalla en los documentos “security.conf” y “apache2.conf” que se copian en el contenedor de Docker.

```
FROM php:8.2.12-apache  
RUN docker-php-ext-install mysqli  
RUN a2enmod headers  
RUN service apache2 restart  
RUN usermod -u 1000 www-data  
RUN groupmod -g 1000 www-data  
COPY security.conf /etc/apache2/conf-enabled  
COPY apache2.conf /etc/apache2/
```

5. Resumen de seguridad implementada en el sistema

En este capítulo se resumen por tipo de vulnerabilidad aquellas frente a las que se ha protegido al sistema.

■ Control de acceso

- Denegación de acceso (mediante interfaz gráfica y URL) al formulario de modificación de datos de usuario si no se tiene iniciada la sesión.
- Denegación de acceso (mediante interfaz gráfica y URL) al formulario de registro e inicio de sesión si la sesión está iniciada.
- Denegación de acceso mediante URL al directorio *openssl* en el que se almacena la clave de cifrado.
- Uso del sistema reCAPTCHA de Google para enviar los formularios.

■ Criptografía

- Cifrado de datos personales *at rest* usando el algoritmo por bloques AES-256.
- No se almacenan las contraseñas de los usuarios del sistema, sino el resumen *hash* de su contraseña combinada con el valor de su *salt*.
- Los datos cifrados no se descifran automáticamente al consultarlos a la base datos; es decir, al hacer una consulta se obtienen cifrados y a la hora de mostrarlos en la interfaz gráfica es cuando se descifran.
- No se mandan datos críticos a través de URL, por lo que para enviar información al servidor en lugar de utilizar el método GET se utiliza el método POST, que envía la información a través de la petición HTTP.

■ Inyección

- Consultas SQL parametrizadas para evitar ataques de inyección.

■ Diseño inseguro

- Uso de APIs para las modificaciones en la base de datos.
- Uso de estilos y scripts en línea deshabilitados mediante la CSP. Por ello, se han implementado los estilos y los scripts en ficheros independientes.

■ Configuración de seguridad

- Debido a la realización en Docker y a que se trata de un proyecto formativo no se ha realizado: bloquear los puertos no necesarios y dejar, por ejemplo, únicamente el 80 (HTTP) y el 443 (HTTPS).
- Cabeceras de respuesta “X-Powered-By”, “Server” y “X-Content-Type-Options”.
- Cabecera Anti-clickjacking.
- Cabeceras CSP.
- Configuración “HttpOnly” y “SameSite” en cookies.
- Tokens anti-CSRF.

■ Componentes vulnerables y obsoletos

- Se ha actualizado la versión PHP a 8.2.12 para poder soportar las cabeceras de seguridad en Apache.
- Se utiliza la versión 2 de reCAPTCHA debido a que la 3 es aún *beta* y podría ser vulnerable.
- Se impide crear una entrada de un horóscopo sin nombre.

■ Fallos identificación y autenticación

- Se impiden los ataques de fuerza bruta gracias al reCAPTCHA.
- No se almacenan las contraseñas sino su *hash* con el *salt*.
- Se ha establecido una política de tamaño y complejidad de contraseñas.
- No se exponen identificadores de sesión en la URL.
- Se invalidan las sesiones después de cerrar sesión.

■ Integridad de datos y software

- La plantilla de Bootstrap ha sido obtenido de la web oficial, con sus debidos certificados por lo que la fuente es de confianza.
- El reCAPTCHA es de Google, empresa de internet popularmente conocida.

■ Monitorización de la seguridad

- Se registran en *logs* los inicios sesión y los intentos con sus respectivos fallos, tanto si son por el token anti-CSRF o por las credenciales.

6. Análisis sobre las implementaciones de seguridad

Se exponen los resultados obtenidos con la herramienta ZAP después de subsanar las vulnerabilidades descritas en esta auditoría. Estas son las alertas obtenidas, once en total:

- *Absence of Anti-CSRF Tokens*
- *CSP: Wildcard Directive*
- *Content Security Policy (CSP) Header Not Set*
- Cross-Domain JavaScript Source File Inclusion
- Authentication Request Identified
- Information Disclosure - Sensitive Information in URL
- Information Disclosure - Suspicious Comments
- Loosely Scoped Cookie
- Session Management Response Identified
- User Agent Fuzzer
- User Controllable HTML Element Attribute (Potential XSS)

A simple vista puede parecer que no hemos conseguido grandes cambios, sin embargo, hay que analizar el detalle:

- Los token anti-CSRF se comprueban buscando aquellos conocidos, es decir, utilizando alguna librería concreta. Sus nombres se determinan en la línea 106 del código GitHub de ZAP (<https://github.com/zaproxy/zap-extensions/blob/main/addOns/pscanrules/src/main/java/org/zaproxy/zap/extension/pscanrules/CsrfCountermeasuresScanRule.java>):
`List<String> tokenNames = extAntiCSRF.getAntiCsrfTokenNames();`

Consecuentemente, nuestra implementación que se ha realizado siguiendo los enlaces bibliográficos del apartado ocho es correcta.

- Nos indica que la directiva CSP es agresiva debido a la inclusión de enlaces como “cdn.jsdelivr.net” para la plantilla HTML de Bootstrap. Sin embargo, esta fuente es de confianza con sus respectivos certificados de autenticación. Lo mismo ocurre con la inclusión de una fuente JavaScript (<https://www.google.com/recaptcha/api.js>), en este caso, para el reCAPTCHA.

- Indica la ausencia de cabecera CSP para la URL “http://localhost:81/robots.txt”. Como ese archivo no existe, es una alerta sin importancia.

- El resto de alertas son informativas, y sólo hay una nueva respecto a los análisis previos: “User Agent Fuzzer”.
Se trata de una alerta meramente informativa que comprueba si hay diferencias en la respuesta en función del agente de usuario modificado (sitios móviles, acceso como rastreador de motor de búsqueda, ...). Compara el código de estado de la respuesta y el código hash del cuerpo de la respuesta con la respuesta original. Tampoco es una alerta de la que debamos preocuparnos.

7. Conclusiones

La comparativa de ambas auditorías muestran una clara imagen del incremento de la seguridad en el sistema web. Sin embargo, resulta fundamental realizar un análisis pormenorizado de los resultados obtenidos por ZAP, puesto que no todas las alertas son realistas, como se ha demostrado en el análisis final. Por ello, una de las principales conclusiones es que el uso de este tipo de herramientas es muy útil puesto que suponen un primer paso para comenzar a atajar posibles vulnerabilidades en el sistema, pero en ningún caso representan la totalidad de fallos de seguridad presentes. Estas correcciones deben ser complementadas con un análisis exhaustivo del código. Por ejemplo, ZAP no detecta fallos criptográficos, y son un tipo de vulnerabilidad muy relevante debido a que pueden ocasionar el incumplimiento de la legislación vigente, entre otras consecuencias.

Por otra parte, ZAP es un software que permite identificar las brechas de seguridad de una manera ética, sin explotarlas; mientras que atacar e intentar penetrar el sistema simulando técnicas de los atacantes podría ofrecernos una mejor perspectiva sobre la seguridad del mismo. Cualquier procedimiento destinado a poner en peligro el sistema es bienvenido para poder detectar las vulnerabilidades y solventarlas eficazmente.

Destacar también que la seguridad no es un estado al que se llegue, sino es un proceso que se desarrolla durante el ciclo de vida del sistema, por lo que los análisis han de ser realizados periódicamente y el sistema actualizado para detener nuevos ataques.

En resumen, es de vital importancia ofrecer un sistema seguro por varios motivos: el rendimiento económico que obtenemos por el servicio o de la empresa para la que trabajamos, las consecuencias legales de incumplir la ley (régimen sancionador), nuestra reputación como profesionales y confianza transmitimos, etc.

¿Cómo nos van a afectar las brechas de seguridad en nuestros sistemas? ¿Y al rendimiento económico? ¿Y a nuestro puesto de trabajo? ¿Qué nos sucede si no cumplimos con la ley? ¿Cómo recuperamos la reputación y la confianza después de producirse fallos de seguridad? ¿Cómo lo demostramos? Son preguntas que nos debemos hacer para reflexionar sobre las decisiones que tomaremos a la hora de diseñar e implementar un sistema informático seguro.

8. Bibliografía

Los enlaces bibliográficos utilizados se agrupan por cada apartado.

1. Tokens Anti-CSRF:

<https://www.zaproxy.org/docs/alerts/10202/>
https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
<https://www.invicti.com/blog/web-security/protecting-website-using-anti-csrf-token/>
<https://backtrackacademy.com/articulo/anti-csrf-tokens-para-prevenir-cross-site-request-forgery-csrf>

2. CSP Header:

<https://www.zaproxy.org/docs/alerts/10038-1/>
<https://content-security-policy.com/>
<https://owasp.org/www-community/attacks/xss/>
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>
<https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>
<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/meta>
<https://bluetriangle.com/blog/how-to-implement-a-content-security-policy-csp>

3. Anti-clickjacking header:

<https://www.zaproxy.org/docs/alerts/10020-1/>
<https://owasp.org/www-community/attacks/Clickjacking>
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/frame-ancestors>
https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>

<https://tecadmin.net/configure-x-frame-options-apache/>

4. Cookie No HttpOnly Flag:

<https://owasp.org/www-community/HttpOnly>
<https://probely.com/vulnerabilities/cookie-without-httponly-flag>
<https://www.zaproxy.org/docs/alerts/10010/>
<https://www.php.net/manual/en/function.setcookie.php>

5. Cookie without SameSite Attribute:

<https://www.zaproxy.org/docs/alerts/10054/>
<https://web.dev/articles/samesite-cookies-explained>

<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cookie-same-site>

6. Server Leaks Information via “X-Powered-By” HTTP Response Header Field(s):
<https://www.zaproxy.org/docs/alerts/10037/>
<https://ecylabs.com/blog/2021/06/21/how-to-fix-server-leaks-information-via-x-powered-by/>
<https://www.ibm.com/docs/en/control-desk/7.6.1.x?topic=checklist-vulnerability-server-leaks-information>
7. Server Leaks Version Information via “Server” HTTP Response Header:
<https://www.zaproxy.org/docs/alerts/10036-2/>
<https://cqr.company/web-vulnerabilities/information-leakage-via-http-headers/>
<https://ecylabs.com/blog/2021/05/18/web-server-leaks-version-information/>
<https://scanrepeat.com/web-security-knowledge-base/server-leaks-version-information-via-server-http-response-header-field>
8. X-Content-Type-Options Header Missing:
<https://www.zaproxy.org/docs/alerts/10021/>
<https://www.iothreat.com/blog/x-content-type-options-header-missing>
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options>
9. Authentication Request Identified:
<https://www.zaproxy.org/docs/alerts/10111/>
<https://www.zaproxy.org/docs/desktop/addons/authentication-helper/auth-req-id/>
10. Information Disclosure - Sensitive Information in URL:
<https://www.zaproxy.org/docs/alerts/10024/>
11. Information Disclosure - Suspicious Comments:
<https://www.zaproxy.org/docs/alerts/10027/>
12. Loosely Scoped Cookie:
<https://www.zaproxy.org/docs/alerts/90033/>
<https://ecylabs.com/blog/2021/06/21/loosely-scoped-cookie/>
<https://portswigger.net/kb/issues/00500300>
13. Session Management Response Identified:
<https://www.zaproxy.org/docs/alerts/10112/>
14. User Controllable HTML Element Attribute (Potential XSS):
<https://www.zaproxy.org/docs/alerts/10031/>

<https://ecylabs.com/blog/2021/06/21/user-controllable-html-element-attribute/>
<https://www.iothreat.com/blog/user-controllable-html-element-attribute-potential-xss>

15. Legislación:

<https://www.boe.es/buscar/doc.php?id=DOUE-L-2016-80807>
<https://www.boe.es/buscar/doc.php?id=BOE-A-2018-16673>

16. AES-256-CBC:

<https://www.baeldung.com/java-encryption-iv>
<https://www.boxcryptor.com/es/encryption/>

17. Hash + salt:

<https://codereview.stackexchange.com/questions/92869/php-salt-generator>
<https://www.php.net/manual/es/function.rand.php>
<https://www.php.net/manual/en/function.hash.php>
<https://www.php.net/manual/en/function.password-hash.php>

18. Configuración de la cookies:

<https://www.php.net/manual/en/function.setcookie.php>
<https://support.detectify.com/support/solutions/articles/48001048952>
<https://stackoverflow.com/questions/39750906/php-setcookie-samesite-strict>