

Auditoría sobre sistema web

SGSSI

2022-2023

...

https://github.com/Danelitos/SGSSI/tree/entrega_2

Titulación:

Grado en Informática de Gestión y Sistemas de Información

...

3º Curso(1º Cuatrimestre)

Danel Alonso

Adrián Cuadrón

Oier Arribas

20 de noviembre de 2022

Índice

1. <u>Rotura de control de acceso</u>	3
1.1. Acceder a cualquier página sin haber iniciado sesión	3
1.2. Número de intentos contraseña fallida	3
1.3. CSRF Token	5
2. <u>Fallos criptográficos</u>	6
2.1. Guardar contraseña con hash y sal	6
3. <u>Inyección</u>	6
3.1. Consultas parametrizadas	6
4. <u>Diseño inseguro</u>	7
4.1. Evitar reenvío formulario	7
5. <u>Configuración de seguridad insuficiente</u>	8
6. <u>Componentes vulnerables y obsoletos</u>	9
6.1. Poner una versión fija de los servicios	9
7. <u>Fallos de identificación y autenticación</u>	10
7.1. Al cerrar sesión, destruir la sesión	10
7.2. Contraseñas más seguras	11
7.3. Pasado un tiempo de navegación cerrar la sesión.	12
8. <u>Fallos en la integridad de datos y software</u>	13
9. <u>Fallos en la monitorización de la seguridad</u>	13
9.1. Crear logs de inicio de sesión	13
10. <u>Conclusiones</u>	14

1. Rotura de control de acceso

1.1. Acceder a cualquier página sin haber iniciado sesión

Al poner la URL de la ruta, nos dimos cuenta de que podías acceder a todas las páginas sin tener que identificarse. Esto puede ser un gran problema, ya que cualquiera podría acceder al sistema sin ningún problema.

Hemos añadido en cada página que para poder visualizar el contenido tiene que haber iniciado sesión. En este caso, estamos usando el método de “denegación por defecto” y solamente se hace visible la página si se ha iniciado sesión. En caso contrario, se le redirige al usuario a la página de inicio de sesión.

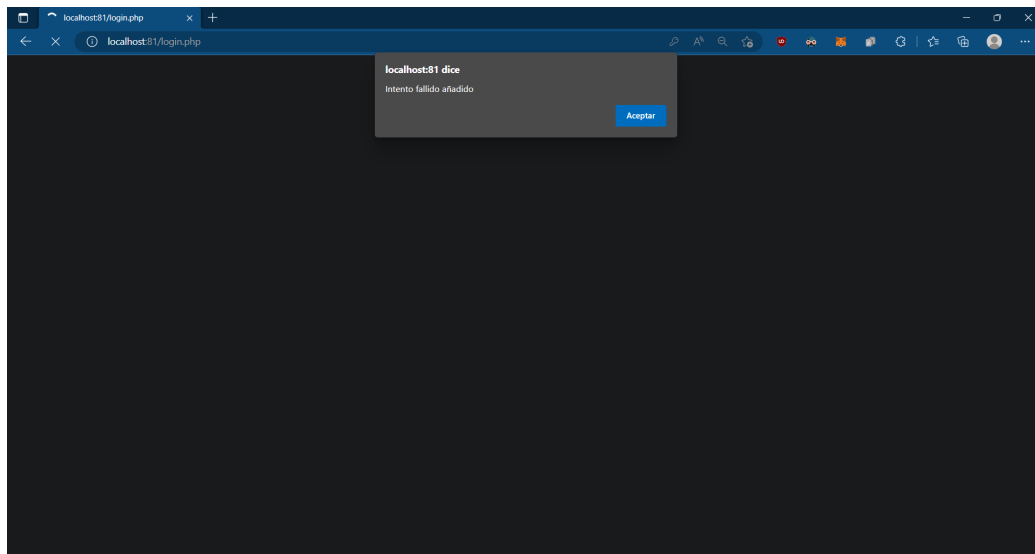
Para ello, hemos usado variables de sesión para saber si un usuario está identificado o no. Con lo cual, si quiere acceder a alguna página que se necesite estar loggeado, será necesario que la variable de sesión no esté vacía.

```
session_start();
if (!isset($_SESSION['miSesion']) && !isset($_SESSION['token'])){
    header("Location:index.php");
}
```

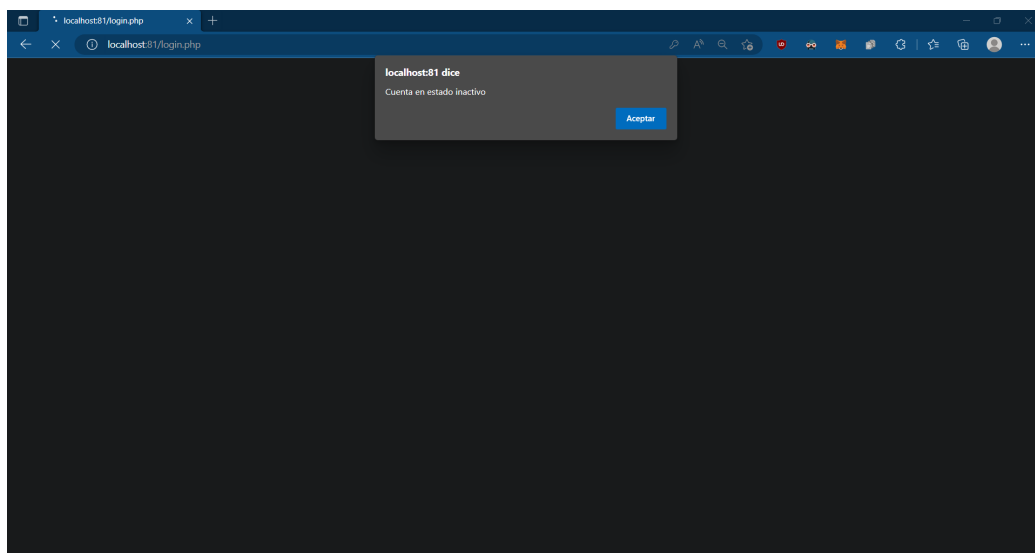
1.2. Número de intentos contraseña fallida

En la base de datos hemos realizado algún cambio para controlar los intentos fallidos de cada usuario. En la tabla de usuario hemos añadido dos nuevos atributos, “IntentosFallidos” y “Estado”.

El atributo de “IntentosFallidos” es un contador que va incrementando cuando el usuario mete mal la contraseña.



Cuando llega a 3 intentos fallidos, la cuenta se desactiva poniendo el “Estado” a inactivo. En este momento la cuenta queda inactiva y no se podrá iniciar sesión, a menos que el administrador le cambie el estado a activo en la base de datos. Gracias a esto conseguimos detectar y evitar ataques por fuerza bruta.



1.3. CSRF Token

Para asegurarse que una acción está realmente siendo llevada a cabo por el usuario en lugar de un tercero, hay que asociarlo con algún tipo de identificador único que puede ser verificado después, llamado token. Para prevenir el ataque, crearemos el token a la hora de hacer el login de la siguiente manera:

```
$_SESSION["token"] = md5(uniqid(mt_rand(), true)); //cuando el inicio es correcto, se crea un TOKEN de sesión
```

Para proteger formularios, se suele incluir el identificador dentro de un campo escondido, siendo enviado con el resto de los datos del formulario:

```
<form class="formulario" method="POST">
  <label>Nombre</label>
  <input class="controles" placeholder="Ingrese el nombre" type="text" minlength="3" name="nombreCoche" /> <br />
  <label>Marca</label>
  <input class="controles" placeholder="Ingrese la marca" type="text" minlength="3" name="marca" /> <br />
  <label>Color</label>
  <input class="controles" placeholder="Ingrese el color" type="text" name="color" /> <br />
  <label>Caballos</label>
  <input class="controles" placeholder="Ingrese los caballos" type="text" pattern="[0-9]{3}" minlength="2" maxleng
  <label>Precio</label>
  <input class="controles" placeholder="Ingrese el precio" type="text" pattern="[0-9]{6}\,[0-9]{2}" minlength="2"
  <input type="hidden" name="csrf" value="<?php echo $_SESSION["token"]; ?>" />
  <?php if(!empty($message)): ?>
  <p> <?= $message ?></p>
  <?php endif; ?>
  <input class="botones" type="submit" value="Añadir Coche" name="botonAñadir" />
  <a href="coches.php"><p>Volver</p></a>
</form>
```

Y para verificar que el token es el mismo, haremos lo siguiente:

```
session_start();
if (!isset($_SESSION['miSesion']) && $_GET["csrf"] == $_SESSION["token"]){
    header("Location:index.php");
}
```

Entonces el atacante para que su ataque salga con éxito tendrá que adivinar el token aleatorio.

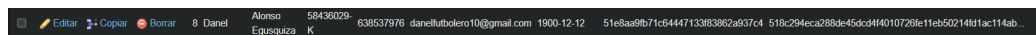
2. Fallos criptográficos

2.1. Guardar contraseña con hash y sal

Cuando el usuario mete la nueva contraseña, o la cambia, creamos una "sal" que identifica esa contraseña. Después esa "sal" la pasamos a hash y junto con la contraseña que ha metido el usuario, lo pasamos de nuevo a hash. Almacenamos por una parte la sal, y por otra la contraseña con la sal en hash.

```
$password = $_POST['password'];  
$salt = md5(uniqid(rand(), true)); // O incluso mejor si tuviese mayúsculas, minúsculas, caracteres especiales...  
$contraseña = hash('sha512', $salt.$password); // Puede ponerse delante o detrás, es igual
```

Al iniciar sesión, cuando el usuario mete esa contraseña se comprueba que al pasar a hash la contraseña metida y junto con la sal del usuario, es la misma que la contraseña almacenada en la base de datos. Así quedaría la contraseña con la sal en la base de datos:



A screenshot of a terminal window. The top bar shows icons for editing, copying, and deleting, along with the name 'Daniel Egusquiza' and a session ID '58436029'. The main area shows a command prompt with a long alphanumeric string: '639537976 danielutbolero10@gmail.com 1900-12-12 51e8a9fb71c6444713383962a937c4 518c294eca288de45dcd4f4010726e11eb50214d1ac114ab...'

3. Inyección

3.1. Consultas parametrizadas

En nuestro sistema, ya teníamos arreglado este fallo de seguridad desde la primera entrega. En nuestra página web, como a la hora de enviar los formularios tenemos las consultas parametrizadas, no es posible hacer "SQL Injection". Sin embargo, hemos hecho varios intentos de SQL Injection para comprobar dicha vulnerabilidad.

Los parámetros son valores que se añaden a la consulta al ejecutarla y son analizadas de forma literal, es decir, si la variable hace referencia a nombres de columnas, el interprete se asegura de que los parámetros son nombre de columnas antes de añadirlas a la consulta y ejecutarlas.

Gracias a tener las consultas parametrizadas, evitamos que cualquier usuario pueda meter parte de código SQL en los campos de los formularios y el programa pueda interpretar lo que haya escrito directamente.

```

if (!empty($_POST['nombreCoche']) && !empty($_POST['marca']) && !empty($_POST['color']) && !empty($_POST['caballos']) && !empty($_POST['precio'])) {
    $nombreCoche=$_POST['nombreCoche'];
    $marca=$_POST['marca'];
    $color=$_POST['color'];
    $caballos=$_POST['caballos'];
    $precio=$_POST['precio'];

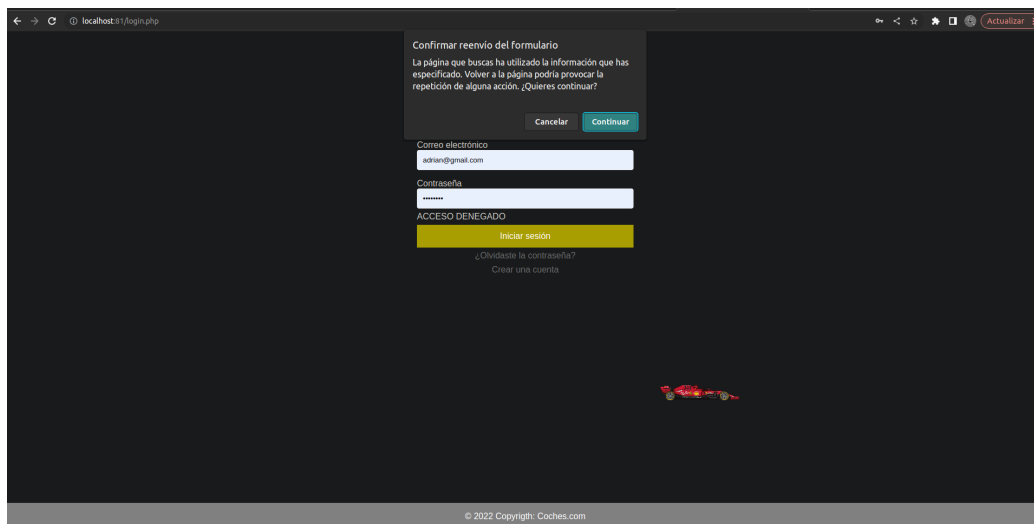
    $sql = "INSERT INTO `coches` (Nombre,Marca,Color,Caballos,Precio) VALUES (?, ?, ?, ?, ?)";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("sssis", $nombreCoche,$marca,$color,$caballos,$precio);
    if ($stmt->execute()) {
        $message=<div class="alert alert-danger">Se ha añadido con éxito el coche</div>;
    } else {
        $message=<div class="alert alert-danger">No se ha podido añadir el coche</div>;
    }
}

```

4. Diseño inseguro

4.1. Evitar reenvío formulario

Cuando enviábamos alguno de los formularios en la pagina, nos dimos cuenta de que si recargabas la pagina se volvía a enviar el formulario. Se puede apreciar en la siguiente foto:



Por este motivo, podríamos tener duplicadas las cosas en la base de datos, lo que seria un fallo de diseño y nos complicaría las cosas. Para evitar esto hemos añadido una función en javascript que evita este reenvío de los datos. Esta función lo que hace es, que cuando detecta un cambio en la ventana(actualizarla), redirige la pagina a una ruta(la misma en este caso, y no se puede reenviar el formulario de nuevo). Lo hemos añadido en los formularios de iniciar sesión y de registro.

5. Configuración de seguridad insuficiente

Este apartado mas que ser cuestión de alterar el código añadiendo nuevas funciones adicionales que mejoren la seguridad de tanto la pagina web como la base de datos hay que encararlo a base de logística.

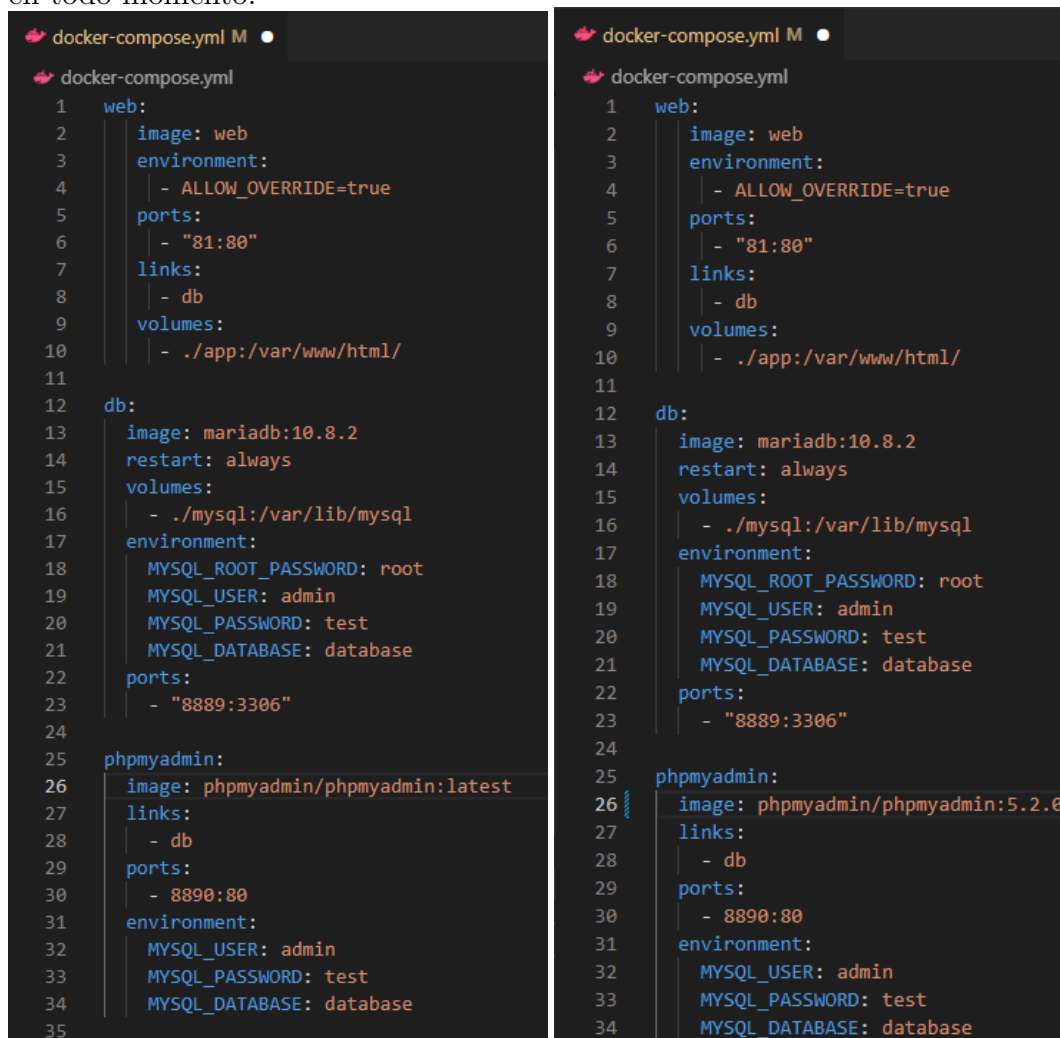
- No existe ninguna cuenta por defecto que se haya usado para hacer pruebas. Como puede ser `usr:admin psw:test`
- No hay ninguna función codificada que no se use para alguna finalidad y que por lo tanto, no sea realmente necesaria.
- No hemos dejado que se desvele ninguna información legible que se pueda filtrar por vía de los errores de la pagina.

6. Componentes vulnerables y obsoletos

6.1. Poner una versión fija de los servicios

No se saben exactamente todas las versiones de los componentes (Siempre se usa "latest"). Esto incluye componentes que se usan directamente y las dependencias indirectas.

En nuestro caso, como podemos ver en el archivo docker-compose.yml usamos el servicio phpmyadmin con la versión "latest". Lo que hemos hecho ha sido cambiar esta versión por 5.2.0, para saber que versión estamos utilizando en todo momento.



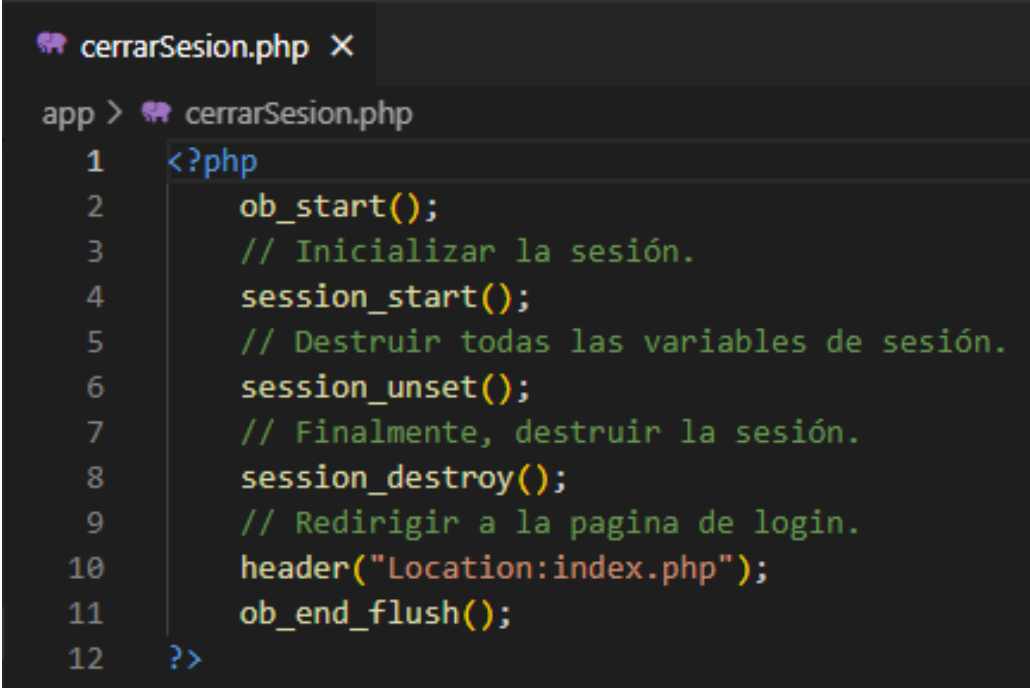
```
docker-compose.yml M
docker-compose.yml
1 web:
2   image: web
3   environment:
4     - ALLOW_OVERRIDE=true
5   ports:
6     - "81:80"
7   links:
8     - db
9   volumes:
10    - ./app:/var/www/html/
11
12 db:
13   image: mariadb:10.8.2
14   restart: always
15   volumes:
16     - ./mysql:/var/lib/mysql
17   environment:
18     MYSQL_ROOT_PASSWORD: root
19     MYSQL_USER: admin
20     MYSQL_PASSWORD: test
21     MYSQL_DATABASE: database
22   ports:
23     - "8889:3306"
24
25 phpmyadmin:
26   image: phpmyadmin/phpmyadmin:latest
27   links:
28     - db
29   ports:
30     - 8890:80
31   environment:
32     MYSQL_USER: admin
33     MYSQL_PASSWORD: test
34     MYSQL_DATABASE: database
35

docker-compose.yml M
docker-compose.yml
1 web:
2   image: web
3   environment:
4     - ALLOW_OVERRIDE=true
5   ports:
6     - "81:80"
7   links:
8     - db
9   volumes:
10    - ./app:/var/www/html/
11
12 db:
13   image: mariadb:10.8.2
14   restart: always
15   volumes:
16     - ./mysql:/var/lib/mysql
17   environment:
18     MYSQL_ROOT_PASSWORD: root
19     MYSQL_USER: admin
20     MYSQL_PASSWORD: test
21     MYSQL_DATABASE: database
22   ports:
23     - "8889:3306"
24
25 phpmyadmin:
26   image: phpmyadmin/phpmyadmin:5.2.0
27   links:
28     - db
29   ports:
30     - 8890:80
31   environment:
32     MYSQL_USER: admin
33     MYSQL_PASSWORD: test
34     MYSQL_DATABASE: database
```

7. Fallos de identificación y autenticación

7.1. Al cerrar sesión, destruir la sesión

Antes un usuario al cerrar sesión o logout, no invalidábamos la sesión. Ahora una vez que el usuario cierra sesión, eliminamos todo los datos que tengan que ver con la sesión de dicho usuario.

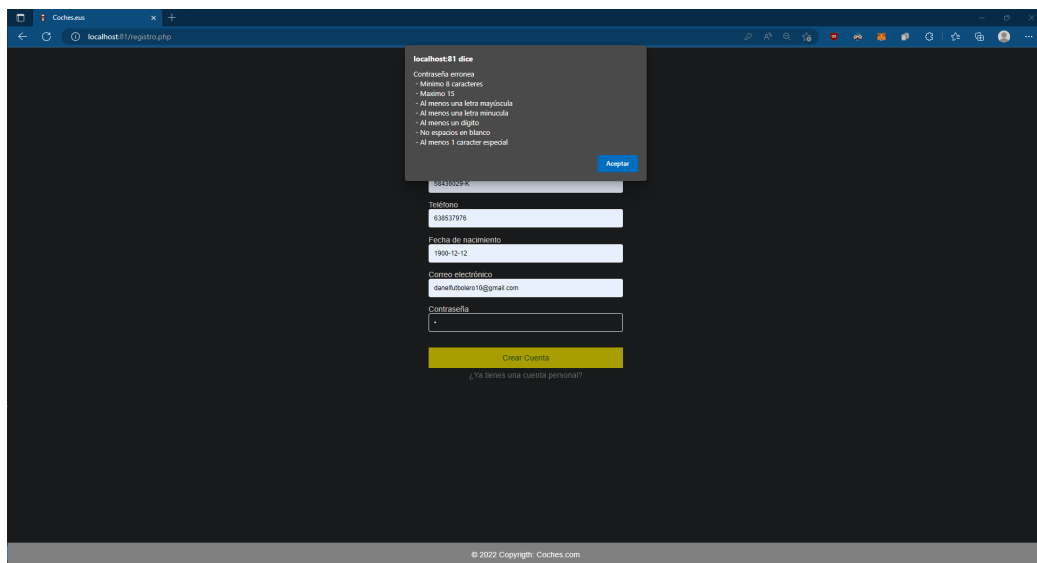


```
cerrarSesion.php X
app > cerrarSesion.php
1  <?php
2      ob_start();
3      // Inicializar la sesión.
4      session_start();
5      // Destruir todas las variables de sesión.
6      session_unset();
7      // Finalmente, destruir la sesión.
8      session_destroy();
9      // Redirigir a la pagina de login.
10     header("Location:index.php");
11     ob_end_flush();
12     ?>
```

7.2. Contraseñas más seguras

Antes podías usar cualquier tipo de contraseña, el único requisito era que tuviese mínimo 8 caracteres.

Para arreglar esta vulnerabilidad lo que hemos hecho, ha sido añadir en el formulario.js una expresión regular, diciendo que la contraseña tenga mínimo 8 caracteres, máximo 15, ningún espacio en blanco y al menos una letra mayúscula, una letra minúscula, un dígito y un carácter especial.



7.3. Pasado un tiempo de navegación cerrar la sesión.

Antes una vez iniciada la sesión podías navegar por la pagina indefinidamente.

Hemos arreglado ese fallo incluyendo un método para que pasado un tiempo de navegación se cierre la sesión automáticamente y te obligue a volver a iniciar sesión. Para ello, en cada interfaz donde necesites tener la sesión iniciada para navegar por ella habrá una función para comprobar si se ha pasado el limite de tiempo. Esta función lo que permite es que tiene un contador para saber si se ha pasado el limite(2min), y si es que si, se cierra la sesión, y se redirige a la pagina de login.php. En este caso hemos puesto 2 minutos por si el profesor quiere probarlo, pero en la realidad pondríamos unos 15-20 minutos para que sea un tiempo suficiente.

```
function timeOut(){
    if(isset($_SESSION['tiempo']) ) {

        //Tiempo en segundos para dar vida a la sesión.
        $inactivo = 118;//2min en este caso.

        //Calculamos tiempo de vida inactivo.
        $vida_session = time() - $_SESSION['tiempo'];

        //Compración para redirigir página
        if($vida_session > $inactivo)
        {
            //Removemos sesión.
            session_unset();
            //Destruimos sesión.
            session_destroy();
            //Redirigimos pagina.
            header("Location: login.php");

            exit();
        }

    } else {
        //Activamos sesion tiempo.
        $_SESSION['tiempo'] = time();
    }
}
```

También en cada interfaz hemos añadido un elemento meta de HTML para que refresque la pagina pasados los 2 minutos, si no el código que se encarga de comprobar el tiempo de sesión y redirigir al login en caso de que fuera necesario solo se ejecutaría la primera vez que se entre a la pagina y cuando el usuario la refresque manualmente. Gracias a esto mejoramos la seguridad en la autenticación.

```
<meta http-equiv="Refresh" content="120">
```

8. Fallos en la integridad de datos y software

Para poder asegurar que los datos son íntegros y que viajan de manera segura, necesitaríamos obtener una certificación de servidor SSL. Como el sistema esta alojado en localhost, no hemos visto necesaria esta parte, pero si en algún casual el sistema lo alojamos en un servidor como Google Cloud, sería necesario tener un certificado valido firmado por una CA. Esto se podría conseguir en paginas como Let's encrypt.

9. Fallos en la monitorización de la seguridad

9.1. Crear logs de inicio de sesión

Cuando cualquier usuario intenta iniciar sesión en la pagina, se añade un nuevo log en el sistema, ya sea el inicio de sesión correcto o fallido. Para ello, hemos creado una nueva carpeta dentro de la aplicación llamada log y dentro hemos creado un archivo php llamado gestionLog.php. En este ultimo tenemos una función que lo que hace es meter dentro del log los datos necesarios. Por otro lado, dentro de la carpeta se genera un archivo log.txt automáticamente. Para que tenga permisos de escritura, podria ser posible que nos pida que necesitamos darle permisos a las carpeta de SGSSI, app y log que están por encima del archivo.

```
$ chmod 777 SQSSI $ chmod 777 SGSSI/app  
$ chmod 777 SGSSI/app/log
```

En nuestro caso, cada fila del log.txt tiene la siguiente estructura:

[FechaHora] [IP cliente] [correo] [descripción]

Ejemplo:

```
2022-11-17 14:53:55 172.17.0.1 adrian@gmail.com Inicio de sesion correcto  
2022-11-17 14:54:28 172.17.0.1 adrian@gmail.com Inicio de sesion fallido
```

Para que no haya problema de conflictos entre los log de los diferentes desarrolladores, tenemos que añadir este archivo(log.txt) al .gitignore, para que cuando hagamos un push a github no se suba el archivo.

10. Conclusiones

Nos hemos dado cuenta de que la pagina web que teníamos era muy insegura y de la gran cantidad de vulnerabilidades que hay que cubrir en una, viendo también los costes que debe tener cubrir todas estas eficazmente.

Sin embargo estos costes son justificados sobre todo en las grandes empresas donde una perdida de información confidencial puede ser catastrófica.