

Diseño de una aplicación web con SQLITE e implementación de medidas de securización de entrada a formularios



19/01/2024

M03 – Marc Ollé Pastor

Índice

Introducción y objetivos	2
Guía de despliegue de la aplicación	3
Pruebas de vulnerabilidad	3
SQL Injection	3
XSS Injection.....	6
HTML Injection	13
Solución a las vulnerabilidades.....	17
SQL Injection	17
XSS.....	20
HTML Injection	22
Webgrafía	23

Introducción y objetivos

Vamos a crear una página web mediante el framework ExpressJS. La página tratará sobre música clásica. A parte de la pasión que yo tengo por la música, otro motivo por el cual selecciono este tema, es porque muchas páginas que hablan sobre este tipo de música están totalmente desfasadas. Son páginas sin ningún tipo de mantenimiento y que además fueron creadas hace mucho tiempo por lo que están totalmente anticuadas.

Vamos a mostrar ciertas vulnerabilidades web que son muy fáciles de ser solucionadas, que son también muy fáciles de que se nos olviden y que además son altamente peligrosas si no se toman en consideración porque se pueden hacer ciertas acciones ilegales si no se tiene una página en condiciones.

Esta página se va a plantear de una manera diferente. La empresa ClassicalFM ha lanzado una nueva iniciativa. Por tal de expandir más el conocimiento que disponen los oyentes de este estilo de música, esta empresa ha decidido de lanzar una página web la cual los usuarios van a depositar los conocimientos que tienen sobre este estilo de música. Para poder acceder a esta página deberán pagar una membresía mensual, la cual garantizará el acceso de estos para poder depositar sus conocimientos. Dicha membresía tendrá un coste de 13,99 euros al mes. Esta incluye acceso a la página ClassicalFM para poder ver sus artículos y también a la página para poder depositar sus conocimientos.

Guía de despliegue de la aplicación

Para poder iniciar esta aplicación se deberá primeramente abrir un cmd o el propio visual studio. Una vez estemos dentro de la carpeta del proyecto la primera cosa a realizar es ejecutar el siguiente comando (Esto en el supuesto caso que no exista la carpeta node_modules):

```
npm install
```

A continuación, ejecutaremos el siguiente comando para inicializar la aplicación

```
npm start
```

Se puede dar el caso en el que nos dé un error al iniciar la aplicación debido a que el puerto está en uso. En este caso el puerto que se usará será el 8000, tal y como se puede ver a continuación:

```
const express = require('express');
const sqlite3 = require('sqlite3');
const path = require('path');

const app = express();
const port = 8000;
const bodyParser = require('body-parser');
```

Si diese error se cambia el valor de esta variable y de esta manera se usará otro puerto.

También se deberá ejecutar el script SQL el cual contiene las tablas y valores pertinentes por tal de que se pueda ejecutar la aplicación correctamente.

```
sqlite3 database.db < database.sql
```

Pruebas de vulnerabilidad

SQL Injection

Accedemos al sitio web, como se puede observar entramos a esta insertando en el buscador "localhost:[puerto] en este caso 8000].

Login

Iniciar sesión

Si ponemos tanto en el nombre de usuario como en contraseña lo siguiente:

' OR '1'='1

Veremos como podremos acceder al sitio:

Login

Iniciar sesión

Bienvenidos a la música que trasciende el tiempo.

Explora lo mejor de la música clásica y sumérgete en la armonía perfecta.

Sobre la Música Clásica

La música clásica es conocida por su profunda estructura y belleza, desde las composiciones de Bach, Mozart, hasta Beethoven. Cada pieza es un viaje a través de la emoción y la técnica.

Si nos vamos al IDE podemos ver con que usuario y contraseña se ha accedido como a su vez la query que se ha usado:

```
server: en http://localhost:3000
{ username: "' OR '1'='1'", password: "' OR '1'='1'" }
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '' OR '1'='1'
Login Exitoso!
```

Mostramos a continuación el código el cual es vulnerable por SQL Injection

```
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  console.log(req.body);

  // Concatenación insegura: SQL Injection permitido
  const query = `SELECT * FROM users WHERE username = '${username}' AND password = '${password}'`;

  // Ejecución de la consulta con SQL Injection permitido
  db.get(query, (err, row) => {
    if (err) {
      console.error(err.message);
      res.status(500).send('Error en la base de datos');
      console.log(query);
      return;
    }

    if (row) {
      // Si se encuentra el usuario, se considera un login exitoso
      console.log(query)
      res.sendFile(path.join(__dirname, '/views/menu.html'));
      console.log("Login Exitoso!");
    } else {
      // Si no se encuentra el usuario, se rechaza el login
      res.sendFile(path.join(__dirname, '/views/error.html'));
    }
  });
});
```

Con esto queda claro que esta página es vulnerable a un ataque de **SQL Injection**.

XSS Injection

Como bien lo hemos visto en clase, los ataques XSS son un tipo de inyección en el que se inyectan secuencias maliciosas en sitios web. Estos ataques ocurren cuando un atacante utiliza una aplicación web para enviar código malicioso, generalmente en forma de una secuencia de comandos del lado del navegador a un usuario. En este caso vamos a realizar esta inyección desde el backend. La página de ClassicalFM permite introducir el conocimiento que dispone los usuarios sobre los diferentes compositores. Dentro del nav observamos como se disponen de varios enlaces a los cuales el usuario se puede dirigir. En este caso si nos dirigimos a la entrada formulario:



**Bienvenidos a la música
que trasciende el tiempo.**

Explora lo mejor de la música clásica y sumérgete en la armonía perfecta.

Sobre la Música Clásica

La música clásica es conocida por su profunda estructura y belleza, desde las composiciones de Bach, Mozart, hasta Beethoven. Cada pieza es un viaje a través de la emoción y la

Se nos abrirá una página la cual contiene un formulario. El usuario aquí pondrá el nombre del compositor y a continuación la información que disponga de este. Mostramos un ejemplo:

Agrega un Compositor

Nombre del Compositor:

Giuseppe Verdi

Información:

Uno de los compositores más importantes de ópera. Ha compuesto operas como: La Traviata, Rigoletto, Otello, La Forza Del Destino y muchas más

ENVIAR

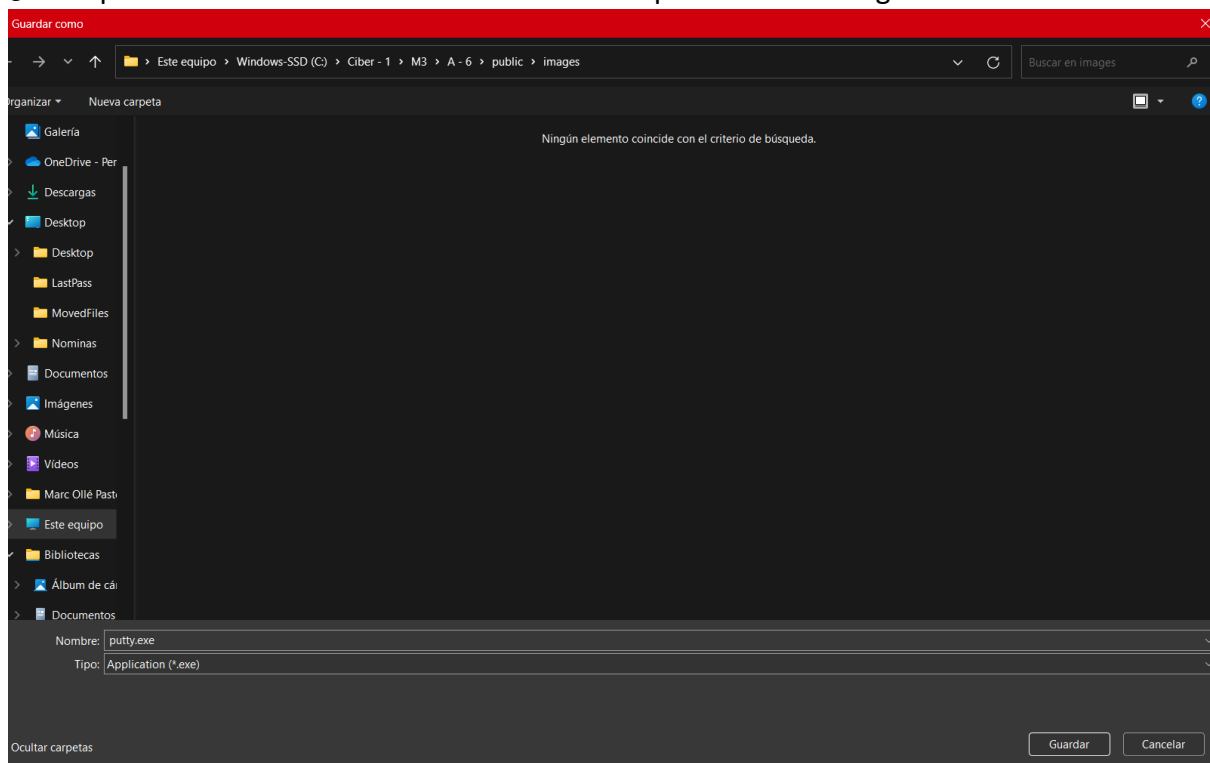
Al clicar al botón de enviar se añadirá a la base de datos los datos que ha introducido el usuario en el formulario y aparecerá la siguiente página, conforme se ha agregado correctamente el registro.

[INICIO](#) [FORMULARIO](#) [CONSULTAR](#) [CONTACTO](#)

SE HA AÑADIDO CORRECTAMENTE TU APOORTE

VOLVER A LA PÁGINA INICIAL

¿Pero qué sucede? Si clicamos el botón veremos que se nos descargará un fichero



En este caso es el ejecutable del Putty pero esto podría ser cualquier fichero. Mostramos a continuación el código:

```
<script>

const boton = document.getElementById("botonVolver");

boton.addEventListener("click", function(){
  console.log("hola");
  const a = document.createElement('a');
  a.href = 'https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe';
  a.download = 'registro.exe';
  document.body.appendChild(a);
  a.click();
  document.body.removeChild(a);
  console.log("hola");
});

setTimeout(function(){
  window.location.href = 'http://localhost:8000/inicio';
}, 9000);

</script>


})
;
```

Como se puede observar el botón tiene un Evento de escucha. Esto es como un trigger, al dar clic al botón se realizará una descarga de un fichero, concretamente el ejecutable del

Putty, pero se puede hacer con cualquier fichero. El propósito es mostrar como al hacer clic se pueden ejecutar cosas. Dicho botón aparentemente hará que se vuelva a la página principal pero también hace una descarga del fichero (en este caso no, pero en un caso real sería un fichero malicioso), a su vez, como se puede apreciar hay una función de Timeout el cual hace que en 9 segundos se redirija a la página inicial por lo que dará la impresión que el botón ha hecho lo que debería hacer.

En futuras ocasiones podemos hacer que la página la cual apunte para descargar el fichero sea un servidor apache que contenga algún archivo malicioso y allí realizar la descarga de este.

A su vez mostramos otro ejemplo de XSS, disponemos de una página de contacto la cual contiene el siguiente formulario:



CONTÁCTANOS

Tu Nombre:

Ej: Juan Pérez

Tu Correo Electrónico:

Ej: ejemplo@correo.com

Tu Mensaje:

Escribe tu mensaje aquí...

Enviar Mensaje

Aparentemente todo está bien, pero el campo del mensaje es vulnerable a un ataque XSS. Si rellenamos los dos primeros campos no ocurrirá nada:



The image shows a contact form titled "CONTÁCTANOS". It has three input fields: "Tu Nombre:" with the value "Carlos", "Tu Correo Electrónico:" with the value "Carlos@gmail.com", and "Tu Mensaje:" with the placeholder text "Escribe tu mensaje aquí...". Below the fields is a black button with the text "Enviar Mensaje".

Pero la problemática viene en el campo del mensaje si introducimos el siguiente código:

```
const messageDiv = document.createElement('div');
messageDiv.textContent = 'Has Sido Hackeado';

Object.assign(messageDiv.style, {
  position: 'fixed',
  top: '50%',
  left: '50%',
  transform: 'translate(-50%, -50%)',
  padding: '20px',
  backgroundColor: 'rgb(0, 0, 0)',
  color: 'green',
  borderRadius: '8px',
  textAlign: 'center',
  boxShadow: '0 4px 8px rgba(0, 0, 0, 0.2)',
  fontFamily: 'Arial, sans-serif',
  fontSize: '16px',
});

document.body.appendChild(messageDiv);
```

Lo introducimos y clicamos al botón de enviar mensaje:

CONTÁCTANOS

Tu Nombre:

Tu Correo Electrónico:

Tu Mensaje:

```
// Crear el contenedor del mensaje
const messageDiv = document.createElement('div');
messageDiv.textContent = 'Has Sido Hackeado';
```

Enviar Mensaje

Veremos como nos aparece un pequeño pop-up:

CONTÁCTANOS

Tu Nombre:

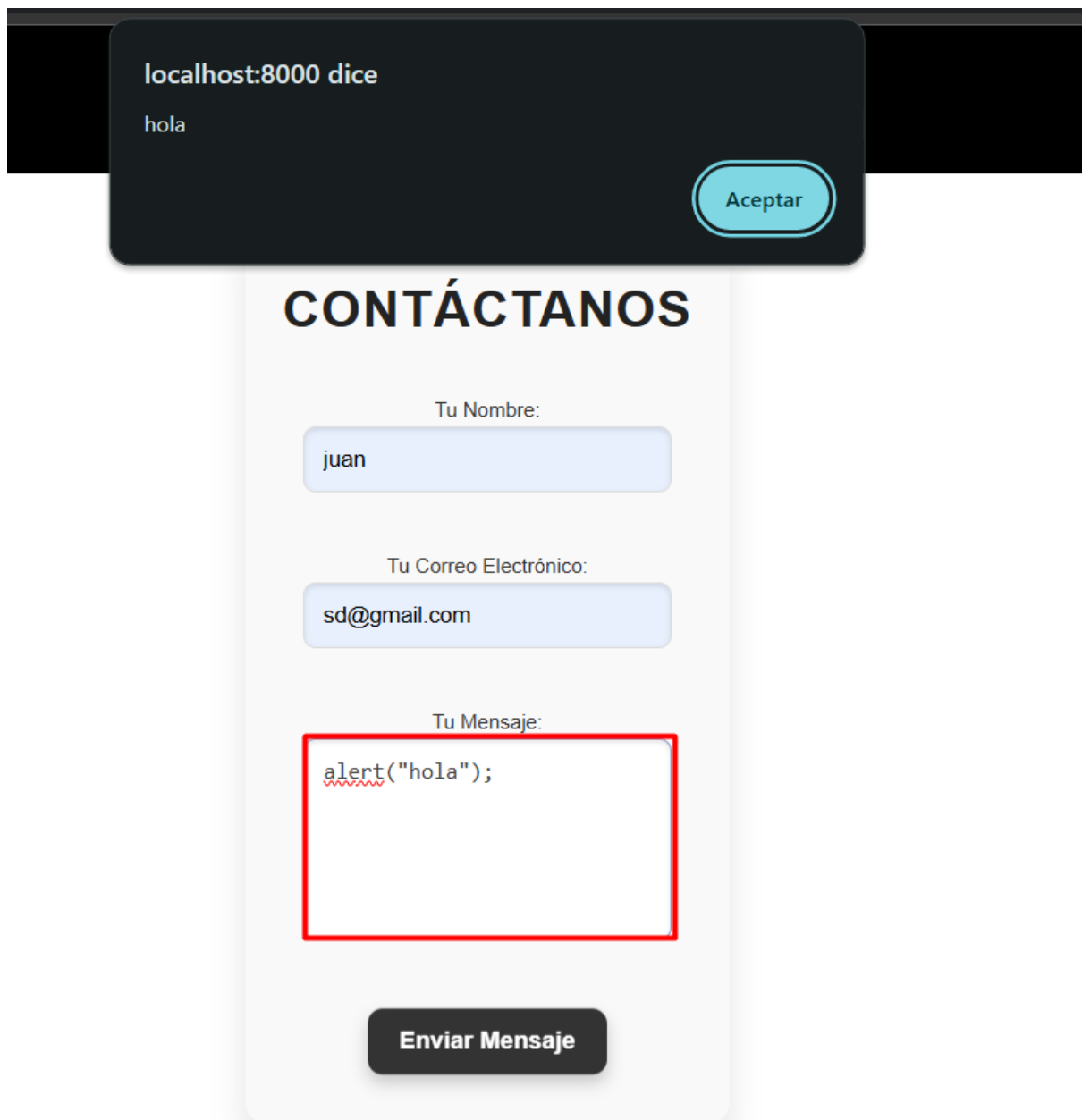
Tu Correo Electrónico:

Has Sido Hackeado

Tu Mensaje:

```
// Crear el contenedor del mensaje
const messageDiv = document.createElement('div');
messageDiv.textContent = 'Has Sido Hackeado';
```

Todo y que no es super XSS, esto se podría considerar un ataque de Cross Site Scripting debido a que el código JS ha sido introducido desde el frontend. Podríamos hacer lo mismo, pero con un simple alert.



The image shows a web interface with a dark header and a light gray contact form. A dark gray alert box is overlaid on top of the form. The alert box contains the text "localhost:8000 dice" and "hola", with an "Aceptar" button. The contact form has the title "CONTÁCTANOS" and three input fields: "Tu Nombre:" with the value "juan", "Tu Correo Electrónico:" with the value "sd@gmail.com", and "Tu Mensaje:" with the value "alert('hola');". The "Tu Mensaje:" field is highlighted with a red border. At the bottom of the form is a black button labeled "Enviar Mensaje".

localhost:8000 dice
hola

Aceptar

CONTÁCTANOS

Tu Nombre:

juan

Tu Correo Electrónico:

sd@gmail.com

Tu Mensaje:

alert("hola");

Enviar Mensaje

Y podemos ver como aparece el alert. El código el cual permite esto es el siguiente:

```
document.addEventListener("DOMContentLoaded", () => {
  const form = document.querySelector("form"); // Asignamos a esta variable la etiqueta de form
  const popup = document.getElementById("popup"); // Asignamos a esta variable el id con el valor popup
  const closePopup = document.getElementById("closePopup"); // Asignamos a esta variable el id con el valor closePopup

  form.addEventListener("submit", (e) => {
    e.preventDefault(); // Evita el envío real del formulario
    // const nombre = form.querySelector("[name='name']").value;
    const mensaje = form.querySelector("[name='mensaje']").value;

    //popup.classList.remove("hidden"); // Muestra el pop-up
    //popup.querySelector(".popup-content").innerHTML = `<p>${nombre}</p><p>${mensaje}</p>`; // Inserta el contenido HTML

    // Ejecutar scripts
    const div = document.createElement("script");
    div.innerHTML = mensaje;
    document.body.appendChild(div);
  });

  closePopup.addEventListener("click", () => {
    popup.classList.add("hidden"); // Oculta el pop-up al darle click
  });
});
```

Con este código permitimos la inyección XSS en la página.

HTML Injection

Disponemos de una página en la aplicación la cual hace una búsqueda filtrada. El usuario introduce el nombre del compositor y al darle en el botón de consultar se efectuará una consulta SQL. Pero no nos adelantemos, mostramos primeramente el formulario.

Consulta de Compositores

[INICIO](#)[FORMULARIO](#)[CONSULTAR](#)[CONTACTO](#)

Busca un Compositor

Nombre del Compositor:

Ej: Wolfgang Amadeus Mozart

Consultar

Si introducimos el nombre de un compositor veremos cómo se hará una búsqueda en la base de datos. Lo que sucederá será que se hará un select con una condición de where. El valor del where será el valor que hemos introducido en el formulario. Por ejemplo, si hacemos la búsqueda del compositor Giuseppe Verdi y clicamos al botón de consultar:

Busca un Compositor

Nombre del Compositor:

Giuseppe Verdi

Consultar

Veremos como muestra las aportaciones que ha puesto el usuario sobre este compositor.

Resultados de la Consulta

INICIOFORMULARIOCONSULTARCONTACTO

NOMBRE	INFORMACIÓN
Giuseppe Verdi	Hola
Giuseppe Verdi	Ha compuesto Nabucco
Giuseppe Verdi	Ha compuesto operas como: La Forza Del Destino, Rigoletto, Otello, Falstaff, La Traviata, Il Trovatore entre muchas más
Giuseppe Verdi	

Giuseppe Verdi

© 2025 Música Clásica. Todos los derechos reservados.

Como se puede ver en la imagen, se muestra la info que se ha aportado sobre el compositor, pero podemos ver como en rojo aparece el nombre del compositor. Pero observemos con detenimiento, allí no debería aparecer el nombre ya que solamente ha habido 3 aportaciones sobre este y además no está dentro de la tabla. ¿Y por qué aparece? Pues bien, esto se debe a que hemos hecho un HTML Injection. Fijémonos en el código, como se observa hay un div que dentro contiene el valor que introduce el usuario en el formulario de consulta dentro de un párrafo.

```

app.get('/mostrarConsulta', (req, res) => {
  if (musicos) {
    const query = `SELECT musico, aportacion FROM musicos WHERE musico = ?`;

    db.all(query, [musicos], (err, resultados) => {
      if (err) {
        console.log(err);
        return res.status(500).send('Error al realizar la consulta');
      }

      console.log(resultados);

      let tablaHTML = '';
      if (resultados.length > 0) {
        tablaHTML = resultados.map(row => {
          console.log(row);
          return `<tr><td>${row.musico}</td><td>${row.aportacion}</td></tr>`;
        }).join('');
      } else {
        tablaHTML = `<tr><td colspan="2">No se encontraron resultados.</td></tr>`;
      }

      res.send(`<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="/stylesheets/resultado.css">
  <title>Resultados de la Consulta</title>
</head>
<body>
  <h1>Resultados de la Consulta</h1>
  <nav class="nav">
    <ul>
      <li><a href="/login">Inicio</a></li>
      <li><a href="/formulario">Formulario</a></li>
      <li><a href="/consultar">Consultar</a></li>
      <li><a href="/contacto">Contacto</a></li>
    </ul>
  </nav>
  <div class="resultados">
    <table>
      <tr>
        <th>Nombre</th>
        <th>Información</th>
      </tr>
      <tr>
        <td colspan="2">${tablaHTML}</td>
      </tr>
    </table>

    <div>
      <p>${musicos}</p>
    </div>
  </div>
  <footer class="footer">

```


Solución a las vulnerabilidades

Ahora que hemos visto las vulnerabilidades que tiene esta página web, procederemos a dar una solución a estas. Como hemos comentado previamente, pero me repito, olvidar estos pequeños detalles en el código puede provocar verdaderos problemas y daños en una página web. Es por ello que es importante cubrir estas vulnerabilidades y tratar de hacer pruebas en la página antes de que esta sea publicada para que ningún cibercriminal pueda aprovecharse y así ponerlo más difícil a los cibercriminales.

SQL Injection

Para solventar esta vulnerabilidad tenemos que implementar sentencias preparadas en nuestro código. Aún con más motivo nos tenemos que reafirmar con lo que hemos comentado previamente, solventar estas vulnerabilidades no es tan complicado es por ello que debemos tener siempre presente que un cibercriminal puede hacer verdaderas atrocidades si no tenemos en cuenta esto. Mostramos a continuación el código el cual soluciona el SQL Injection, en este código a diferencia del anterior usamos sentencias preparadas:

```
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  const query = `SELECT * FROM users WHERE username = ? AND password = ?`;

  db.get(query, [username, password], (err, row) => {
    if (err) {
      console.error(err.message);
      res.status(500).send('Error en la base de datos');
      return;
    }

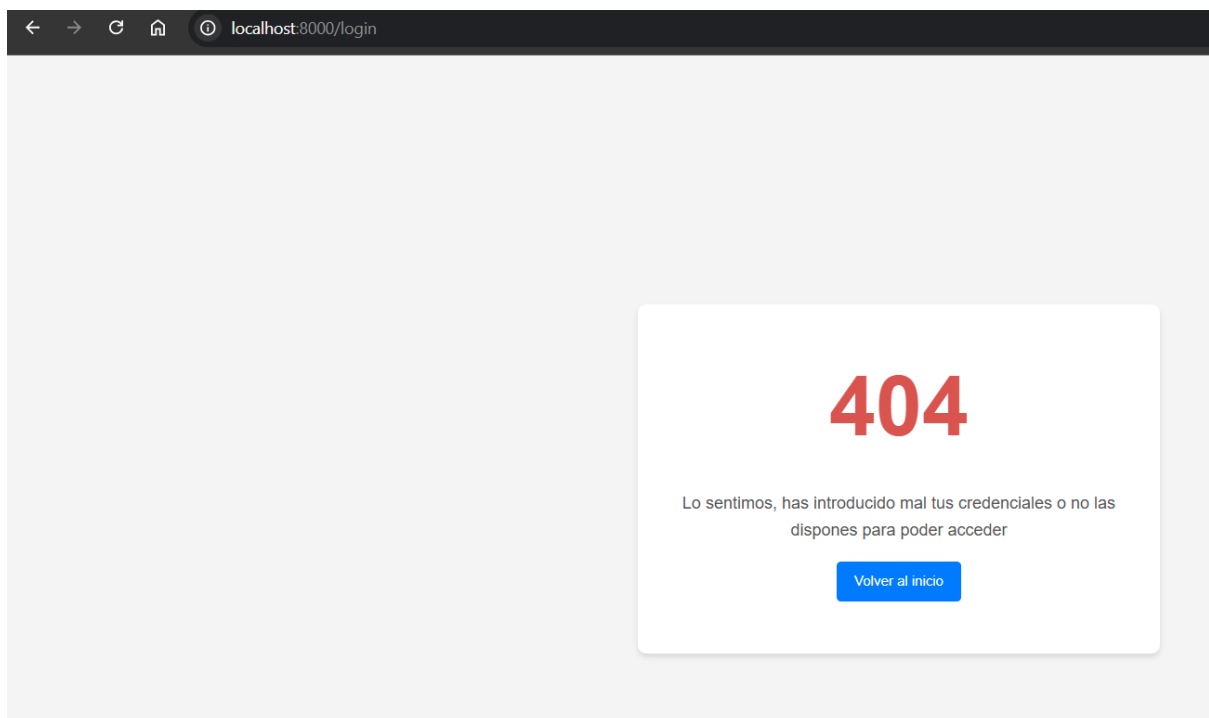
    if (row) {
      res.sendFile(path.join(__dirname, '/views/menu.html'));
      console.log("Login Exitoso!");
    } else {
      // Si no se encuentra el usuario, se rechaza el login
      res.sendFile(path.join(__dirname, '/views/error.html'));
    }
  });
});
```

Como se puede observar se ha creado una página de error la cual saltará si no se encuentra el usuario el cual se ha introducido. A continuación se enseña el proceso de login en la página con el nuevo código:

Se introduce la sentencia para provocar un SQL Injection:

Login

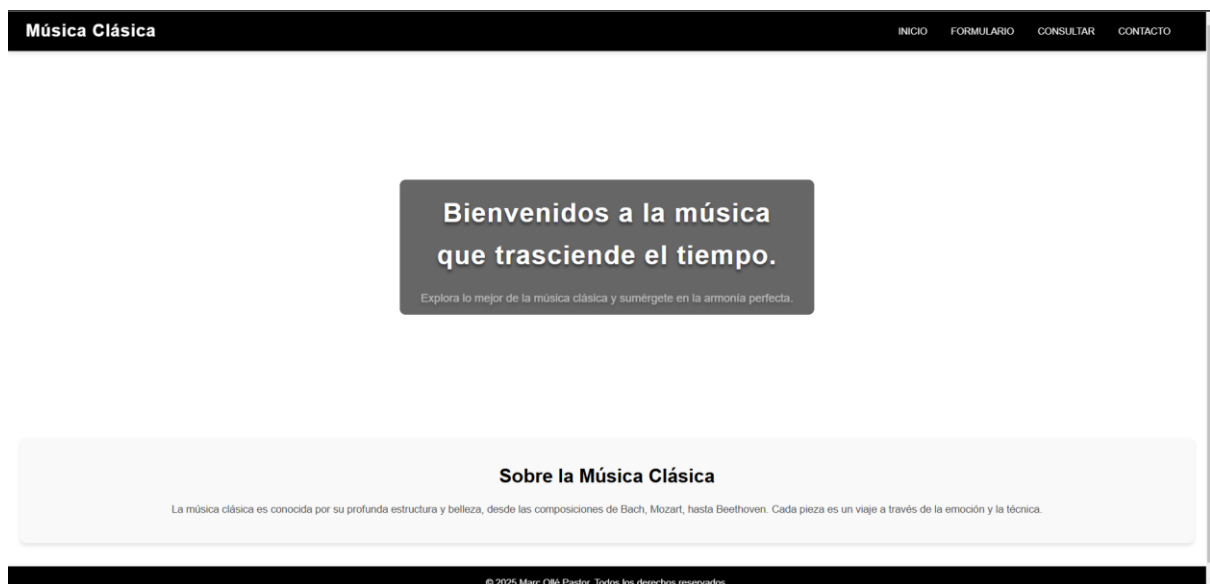
Clicamos a iniciar sesión y nos llevará a la página error.html



Ahora introduciremos unas credenciales válidas, volvemos a la página de login:

Login

Al clicar a iniciar sesión veremos cómo podremos acceder correctamente a la zona de usuario registrado:



XSS

Igual que en el SQL Injection, la solución para este tipo de vulnerabilidad puede ser bastante sencilla. En este caso disponemos de varias soluciones posibles a esta vulnerabilidad.

Mostramos el código el cual es vulnerable a un ataque XSS:

```
document.addEventListener("DOMContentLoaded", () => {
  const form = document.querySelector("form"); // Asignamos a esta variable la etiqueta de form
  const popup = document.getElementById("popup"); // Asignamos a esta variable el id con el valor popup
  const closePopup = document.getElementById("closePopup"); // Asignamos a esta variable el id con el valor closePopup

  form.addEventListener("submit", (e) => {
    e.preventDefault(); // Evita el envío real del formulario
    // const nombre = form.querySelector("[name='nombre']").value;
    const mensaje = form.querySelector("[name='mensaje']").value;

    //popup.classList.remove("hidden"); // Muestra el pop-up
    //popup.querySelector(".popup-content").innerHTML = `<p>${nombre}</p><p>${mensaje}</p>`; // Inserta el contenido HTML

    // Ejecutar scripts
    const div = document.createElement("script");
    div.innerHTML = mensaje;
    document.body.appendChild(div);
  });

  closePopup.addEventListener("click", () => {
    popup.classList.add("hidden"); // Oculta el pop-up al darle click
  });
});
```

Una de las primeras soluciones es no usar el innerHTML, existen métodos más seguros como por ejemplo textContent o createTextNode los cuales no interpretan HTML ni ejecutarán scripts.

A su vez también podemos sanitizar la entrada del usuario antes de que sea procesada. El concepto de sanitizar viene del mundo de la química, su significado sería aplicar calor o químicos para matar a gérmenes en un lugar determinada hasta llegar al punto que estos no supongan un riesgo para la salud. Todo y que esto viene de un mundo diferente, viene de la química, pero el término en el mundo de la informática tiene un significado mucho más parecido de lo que pensamos. En este caso sanitizamos la entrada del usuario para garantizar que no contengan

A continuación, mostramos el código el cual corrige esta vulnerabilidad en nuestra página:

```
document.addEventListener("DOMContentLoaded", () => {
  const form = document.querySelector("form");

  form.addEventListener("submit", (e) => {
    e.preventDefault(); // Evita el envío real del formulario

    const mensaje = form.querySelector("[name='mensaje']").value;

    // Escapa cualquier contenido del usuario para prevenir XSS
    const sanitizedMessage = mensaje.replace(/<|>|'"/g, (char) => {
      const escapeMap = {
        '&': '&amp;',
        '<': '&lt;',
        '>': '&gt;',
        '"': '&quot;',
        "'": '&#39;',
      };
      return escapeMap[char];
    });

    console.log("Mensaje enviado:", sanitizedMessage);
  });
});
```

Por mucho que intentemos ahora poner el código JS no se ejecutará y no sucederá nada.

The screenshot shows a web browser at localhost:8000/contacto. The page has a dark header with the title "Consulta de Compositores" and navigation links: INICIO, FORMULARIO, CONSULTAR, and CONTACTO. The main content area is a light gray box titled "CONTACTANOS". It contains a contact form with three fields: "Tu Nombre" (input type="text" with value "juan"), "Tu Correo Electrónico" (input type="text" with value "s@gmail.com"), and "Tu Mensaje" (text area containing a JavaScript payload: `fontFamily: 'Arial, sans-serif',
fontSize: '16px',
});

// Añadir el div al body
document.body.appendChild(messageDiv);`). Below the text area is a red button labeled "Enviar Mensaje".

Aclarar que en este formulario no existe una lógica después de que se envíe el mensaje, pero claro este formulario previamente era vulnerable a la inyección XSS.

HTML Injection

Para solucionar esta vulnerabilidad es relativamente sencillo ya que el ejemplo de HTML Injection el cual hemos puesto no es muy complicado, pero igualmente no quita que sea una inyección de HTML lo que se ha propuesto. Esta inyección viene por la parte del backend ya que existe un div el cual contiene un párrafo con el valor de una variable como se puede observar:

```
<div>
  <p>${musicos}</p>
</div>
```

La variable `músicos` guarda el valor que le ponemos nosotros en el formulario:

```
app.get('/mostrarConsulta', (req, res) => {
  const musicos = req.query.musico; ←
```

Que al final viene ser lo que pongamos en este input:

Busca un Compositor

Nombre del Compositor:

Consultar

La solución viene a ser muy sencilla es eliminar este div y párrafo totalmente innecesario. Si ahora hacemos una búsqueda en base al compositor Giuseppe Verdi veremos como no aparecerá su nombre fuera de la tabla la cual muestra las aportaciones de los usuarios.

Resultados de la Consulta

INICIO

FORMULARIO

CONSULTAR

CONTACTO

NOMBRE	INFORMACIÓN
Giuseppe Verdi	Hola
Giuseppe Verdi	Ha compuesto Nabucco
Giuseppe Verdi	Ha compuesto operas como: La Forza Del Destino, Rigoletto, Otello, Falstaff, La Traviata, Il Trovatore entre muchas más

© 2025 Música Clásica. Todos los derechos reservados.

Webgrafía

https://developer.mozilla.org/en-US/docs/Glossary/SQL_Injection

https://developer.mozilla.org/es/docs/Glossary/Cross-site_scripting

<https://owasp.org/www-community/attacks/xss/>

<https://expressjs.com/>

<https://expressjs.com/en/starter/installing.html>

<https://www.sqlite.org/cli.html>

<https://learn.snyk.io/lesson/xss/>

<https://www.imperva.com/learn/application-security/html-injection/>

<https://www.invicti.com/learn/html-injection/>

[https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing for HTML Injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/03-Testing_for_HTML_Injection)

<https://www.acunetix.com/blog/web-security-zone/html-injections/>

Data d'entrega i format:

Haureu d'entregar un .zip amb el codi del projecte i el PDF amb la memòria tècnica demanada al pou d'entrega de moodle abans del 14 de gener a les 23.55

Avaluació:

Rúbrica:

Items	10-9	8-7	5	3-4	0-1
Introduccio (memoria) (10%)	S'ajusta el que es demana	-	-	-	no s'aporta introducció ni explicació
Guia del desplegament de l'aplicació web (20%)	Es descriu amb molt detall i precisió els passos i s'aporten imatges de qualitat	Es descriu correctament el procés	Es descriu molt breument el procés o si hi ha errors s'han documentat	L'alumne no documenta els errors i no aconsegueix una app funcional	Altres
Proves de vulnerabilitat amb els formularis (20%)	Es descriu amb molt detall i precisió els passos i s'aporten imatges de qualitat	Es descriu amb molt detall i precisió els passos i s'aporten imatges de qualitat	Es descriu molt breument el procés o si hi ha errors s'han documentat	L'alumne no documenta els errors i no aconsegueix una app funcional	Altres
Implementacio de les mesures i exemples (30%)	Es descriu amb molt detall i precisió els passos i s'aporten imatges de	Es descriu amb molt detall i precisió els passos i s'aporten imatges de	Es descriu molt breument el procés o si hi ha errors s'han documentat	L'alumne no documenta els errors i no aconsegueix una app funcional	Altres

	qualitat	qualitat			
Funcionalitat del codi (20%)	el codi és funcional	-	el codi és mínimament funcional (arrenca però amb errors)	L'alumne no documenta els errors i no aconsegueix una app funcional	altres