



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

SEGURIDAD Y VIRTUALIZACIÓN

Nombre de los Integrantes de equipo:

Edwin López Santiago

Yanet González García

No. Control

21620123

21620273

Tema:

Práctica 4.- Inyección SQL

Docente:

Ing. Osorio Salinas Edward

Carrera:

Ingeniería en Sistemas Computacionales

Grupo: 7US

Tlaxiaco, Oaxaca. A 03 de Septiembre de 2024.



Índice

| | |
|--|-----------|
| Introducción..... | 3 |
| Desarrollo | 4 |
| Práctica 4: Inyección SQL | 4 |
| 1. Crear una base de datos con una tabla que contenga al menos 3 registros. | 4 |
| 2. Crear una aplicación web que permita buscar un registro por su id, nombre o descripción..... | 6 |
| • Esta aplicación debe ser vulnerable a inyección de código, esto significa que, si el usuario ingresa un valor malicioso en el campo de búsqueda, la aplicación debe mostrar información que no debería ser accesible o permitir realizar acciones que no deberían ser posibles..... | 13 |
| 3. Realizar pruebas de inyección de código en la aplicación web..... | 14 |
| 4. Documentar los resultados obtenidos y las posibles acciones que se pueden realizar para prevenir este tipo de vulnerabilidades. | 19 |
| 5. Investiga y describe los siguientes conceptos: | 22 |
| Conclusión..... | 24 |
| Bibliografías..... | 25 |



Introducción

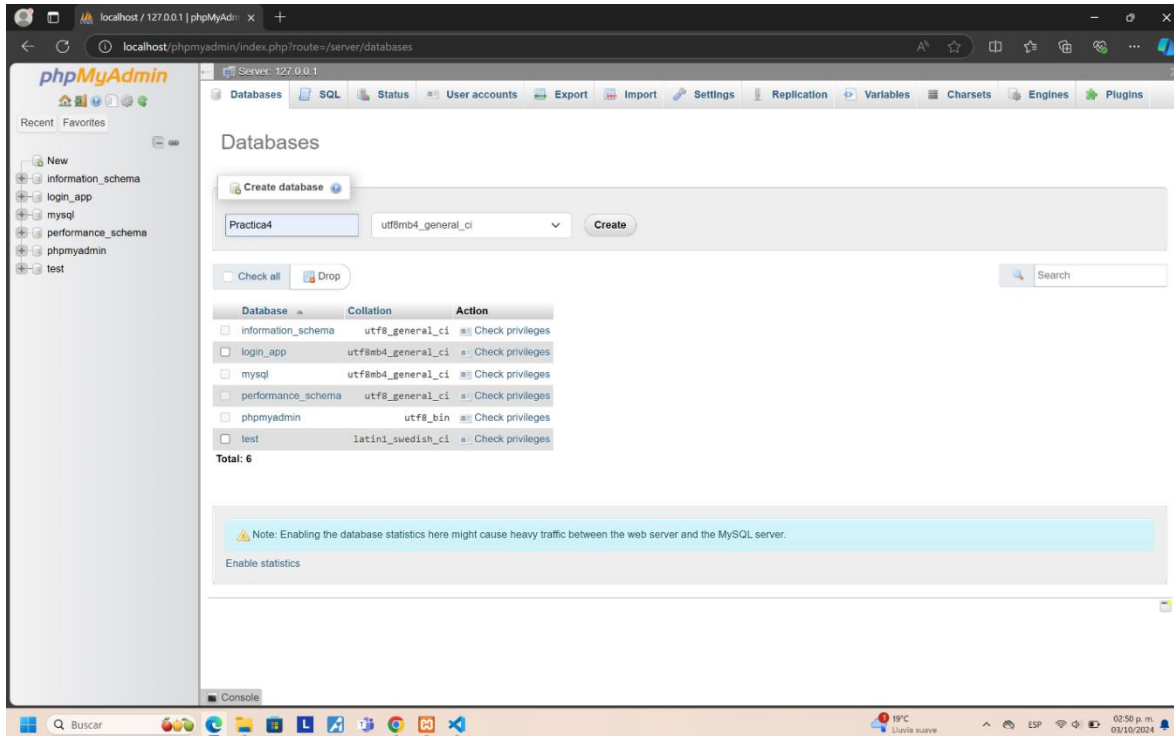
En esta práctica vamos a verificar como se realiza una de las vulnerabilidades más comunes en aplicaciones web: la inyección SQL. El objetivo es crear una base de datos y una aplicación web que permita buscar información en dicha base de datos, pero que sea vulnerable a inyecciones de código. Esto nos permitirá ver cómo los atacantes pueden aprovecharse de una validación inadecuada de las entradas del usuario para manipular la base de datos y obtener información no autorizada o realizar acciones maliciosas, como eliminar datos. Durante la práctica, realizaremos pruebas de inyección en la aplicación y documentaremos los resultados, además de proponer algunas soluciones para evitar este tipo de vulnerabilidades. Este ejercicio es importante porque la inyección SQL sigue siendo una amenaza real en muchas aplicaciones web modernas, y aprender a identificar y prevenir estas vulnerabilidades es fundamental para proteger nuestros sistemas. A continuación, se puede visualizar esta práctica paso a paso.

Desarrollo

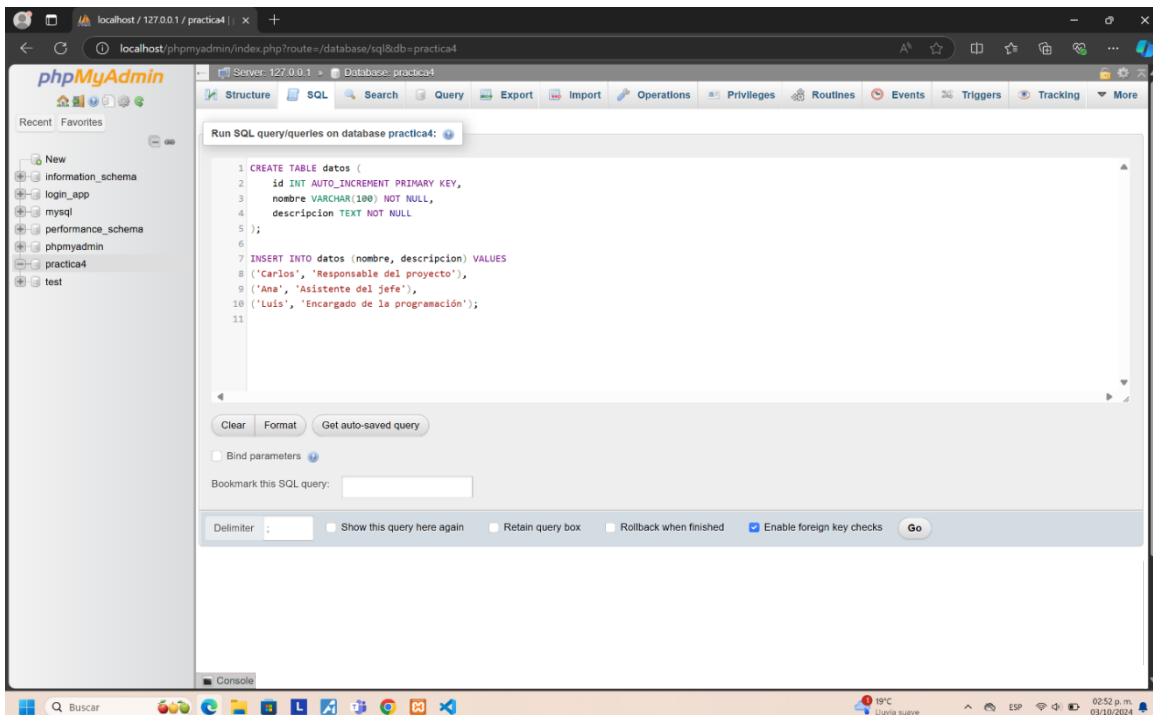
Práctica 4: Inyección SQL

1. Crear una base de datos con una tabla que contenga al menos 3 registros.

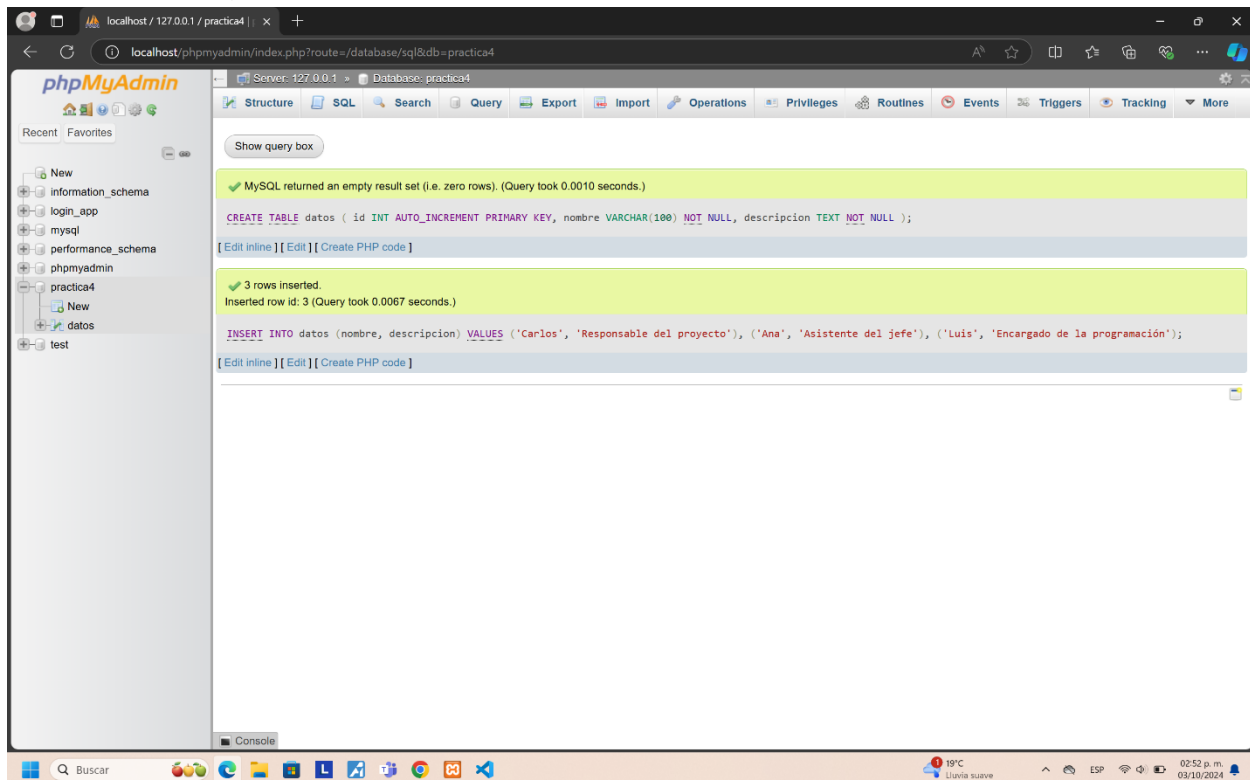
Paso 1. Creación de la base de datos: Al presionar "Create", la base de datos "Practica4" será creada y podremos estructurar los datos que se requiere.



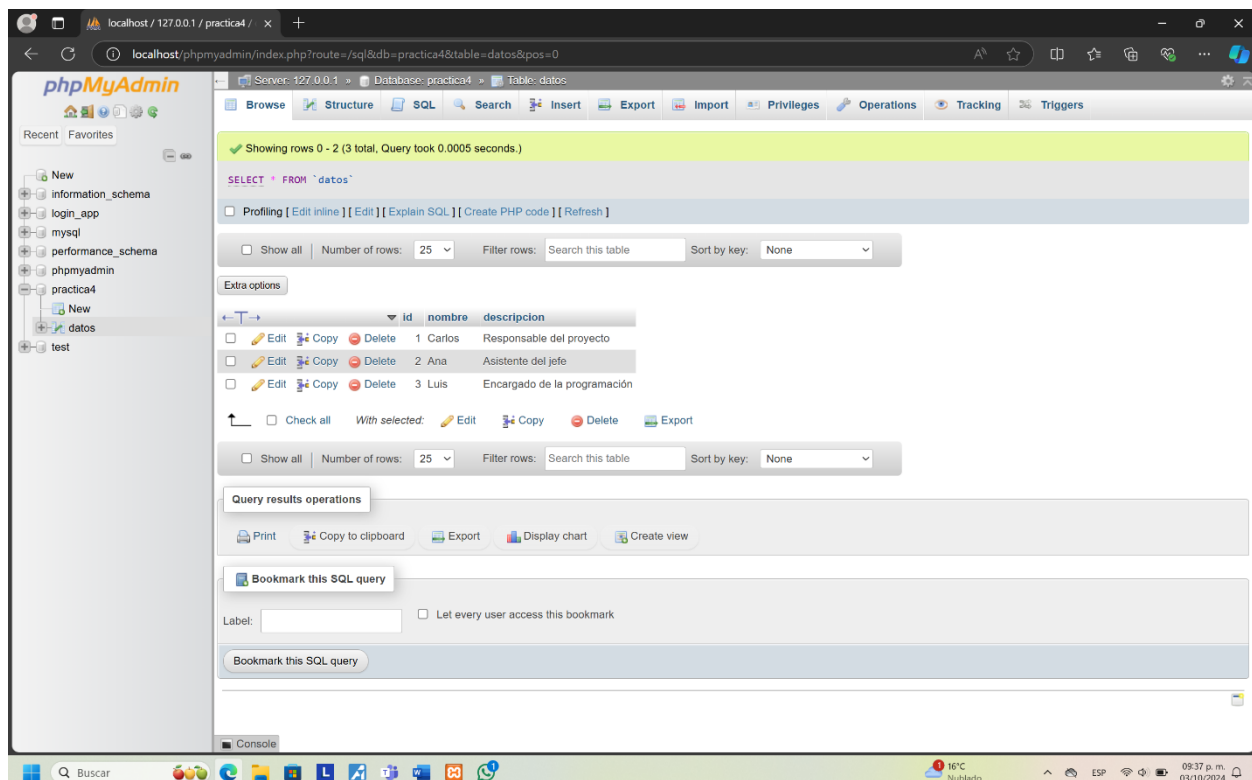
Paso 2. Una vez creado la tabla, creamos una tabla y sus respectivos datos.



Paso 3. Ya que ejecutemos los códigos de creación e inserción de datos nos muestra la siguiente pantalla.

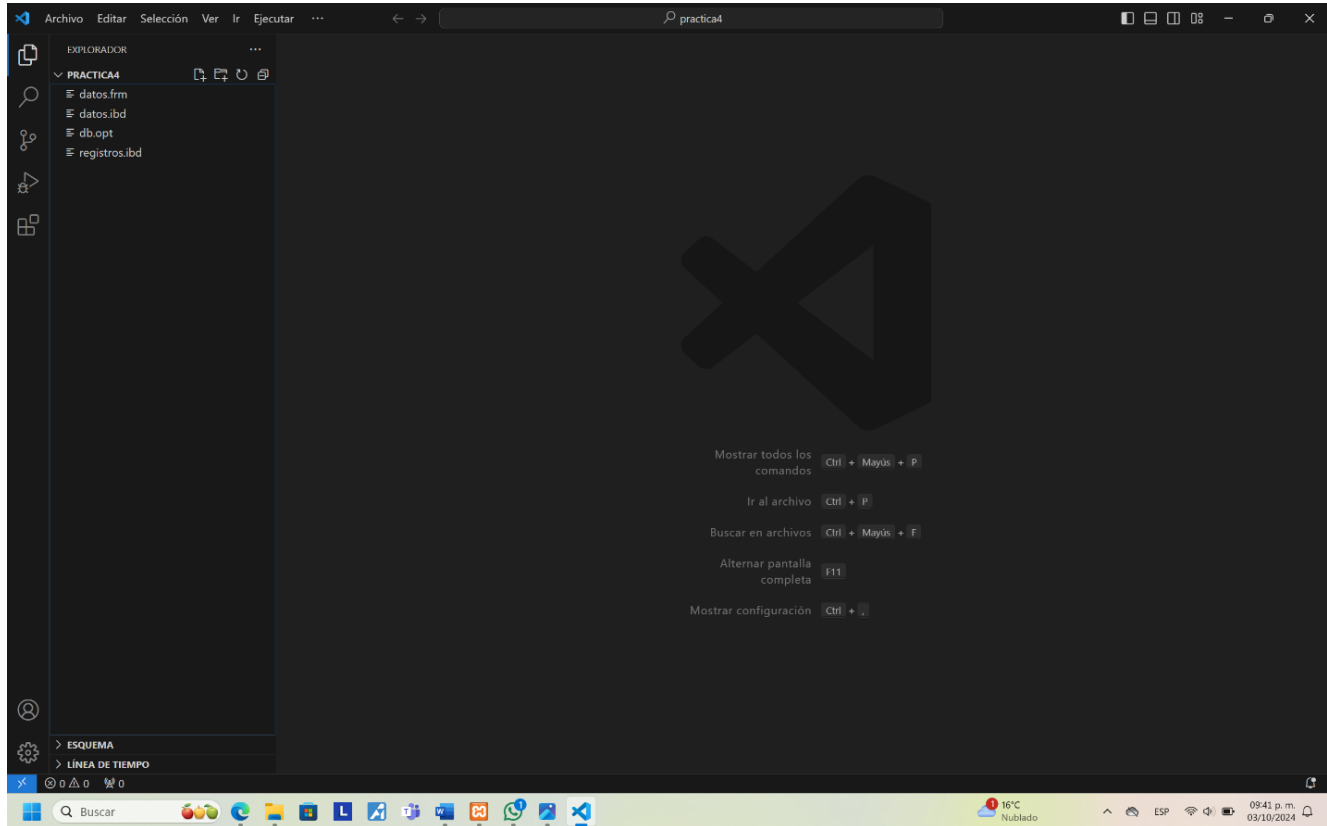


Paso 4. Y así es como se ve nuestra tabla de datos con sus respectivos registros. Este paso muestra cómo los datos pueden ser gestionados una vez que se ha creado la tabla y se han insertado registros. Aquí es posible editar la información, eliminar registros o agregar nuevos.

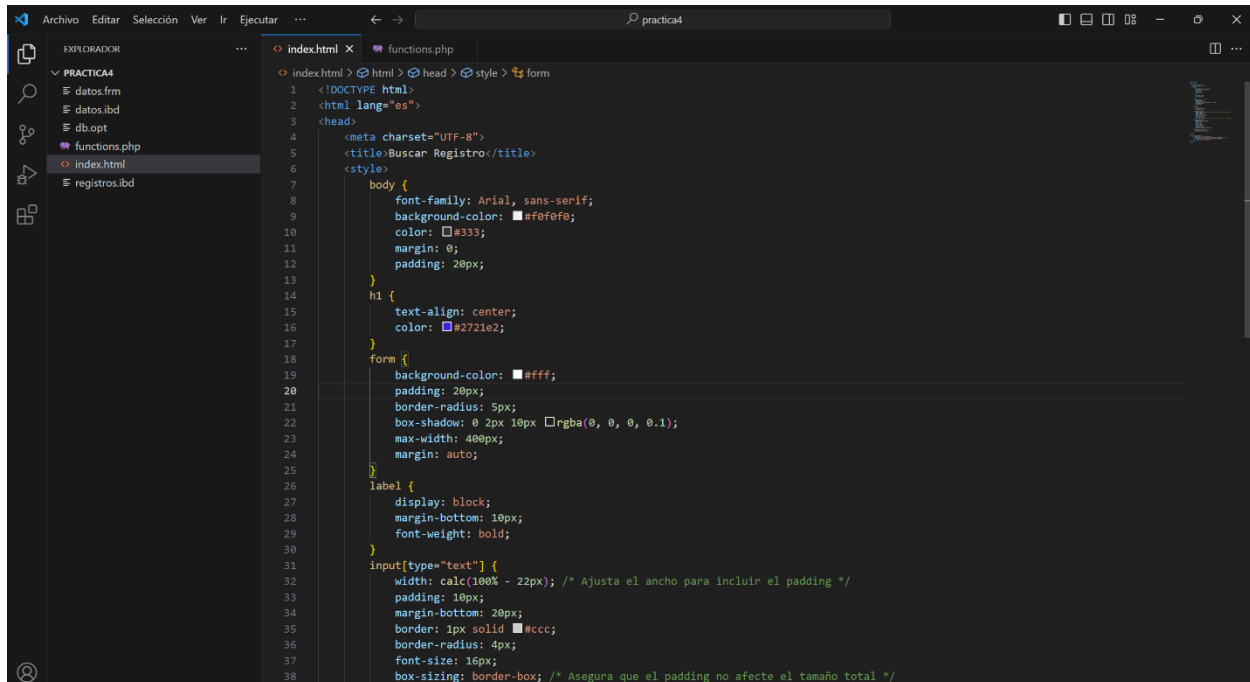


2. Crear una aplicación web que permita buscar un registro por su id, nombre o descripción.

Paso 1. Como primer paso para este punto tenemos que abrir el programa de Visual Code y abrir la carpeta de nuestra base de datos para que en esa misma ubicación tengamos los archivos ejecutables para este paso.

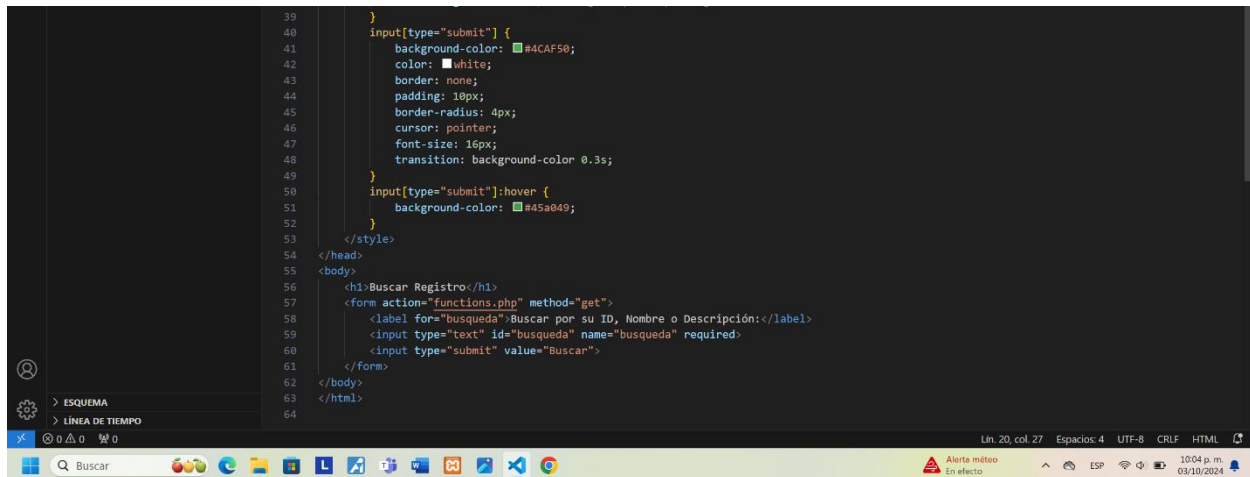


Paso 2. Creación del index de nuestro programa web.



```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <title>Buscar Registro</title>
6 <style>
7   body {
8     font-family: Arial, sans-serif;
9     background-color: #f0f0f0;
10    color: #333;
11    margin: 0;
12    padding: 20px;
13  }
14  h1 {
15    text-align: center;
16    color: #2721e2;
17  }
18  form {
19    background-color: #fff;
20    padding: 20px;
21    border-radius: 5px;
22    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
23    max-width: 400px;
24    margin: auto;
25  }
26  label {
27    display: block;
28    margin-bottom: 10px;
29    font-weight: bold;
30  }
31  input[type="text"] {
32    width: calc(100% - 22px); /* Ajusta el ancho para incluir el padding */
33    padding: 10px;
34    margin-bottom: 20px;
35    border: 1px solid #ccc;
36    border-radius: 4px;
37    font-size: 16px;
38    box-sizing: border-box; /* Asegura que el padding no afecte el tamaño total */
39  }
```

Paso 3. Creación de las funciones para buscar un registro por medio



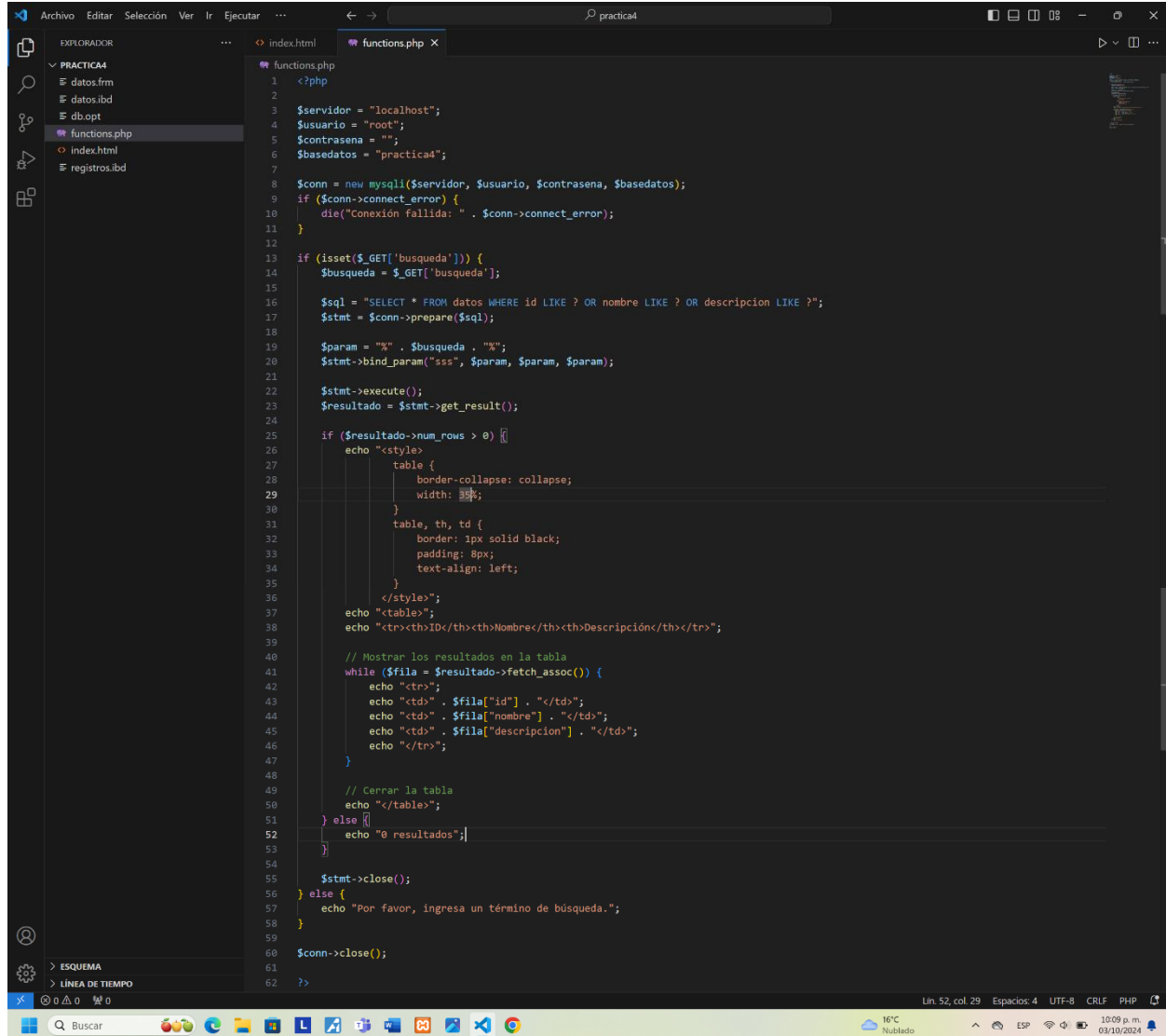
```
39 }
40 input[type="submit"] {
41   background-color: #4CAF50;
42   color: white;
43   border: none;
44   padding: 10px;
45   border-radius: 4px;
46   cursor: pointer;
47   font-size: 16px;
48   transition: background-color 0.3s;
49 }
50 input[type="submit"]:hover {
51   background-color: #45a049;
52 }
53 </style>
54 </head>
55 <body>
56 <h1>Buscar Registro</h1>
57 <form action="functions.php" method="get">
58   <label for="busqueda">Buscar por su ID, Nombre o Descripción:</label>
59   <input type="text" id="busqueda" name="busqueda" required>
60   <input type="submit" value="Buscar">
61 </form>
62 </body>
63 </html>
64
```

Explicación del código utilizado en index.html

Este código crea una página web simple que permite a los usuarios buscar registros mediante un formulario. Está bien estructurado y estilizado para ser atractivo visualmente, y es funcional al enviar las búsquedas a un archivo PHP para su procesamiento.

Paso 4. Creación del archivo functions.php

En este apartado creamos el archivo que se encargara de hacer la búsqueda a través del ID, Nombre o Descripción.



```
1 <?php
2
3 $servidor = "localhost";
4 $usuario = "root";
5 $contrasena = "";
6 $basedatos = "practicad";
7
8 $conn = new mysqli($servidor, $usuario, $contrasena, $basedatos);
9 if ($conn->connect_error) {
10     die("Conexión fallida: " . $conn->connect_error);
11 }
12
13 if (isset($_GET['busqueda'])) {
14     $busqueda = $_GET['busqueda'];
15
16     $sql = "SELECT * FROM datos WHERE id LIKE ? OR nombre LIKE ? OR descripcion LIKE ?";
17     $stmt = $conn->prepare($sql);
18
19     $param = "%" . $busqueda . "%";
20     $stmt->bind_param("sss", $param, $param, $param);
21
22     $stmt->execute();
23     $resultado = $stmt->get_result();
24
25     if ($resultado->num_rows > 0) {
26         echo "<style>
27             table {
28                 border-collapse: collapse;
29                 width: 80%;
30             }
31             table, th, td {
32                 border: 1px solid black;
33                 padding: 8px;
34                 text-align: left;
35             }
36         </style>";
37         echo "<table>";
38         echo "<tr><th>ID</th><th>Nombre</th><th>Descripción</th></tr>";
39
40         // Mostrar los resultados en la tabla
41         while ($fila = $resultado->fetch_assoc()) {
42             echo "<tr>";
43             echo "<td>" . $fila["id"] . "</td>";
44             echo "<td>" . $fila["nombre"] . "</td>";
45             echo "<td>" . $fila["descripcion"] . "</td>";
46             echo "</tr>";
47         }
48
49         // Cerrar la tabla
50         echo "</table>";
51     } else {
52         echo "0 resultados";
53     }
54
55     $stmt->close();
56 } else {
57     echo "Por favor, ingresa un término de búsqueda.";
58 }
59
60 $conn->close();
61
62 ?>
```

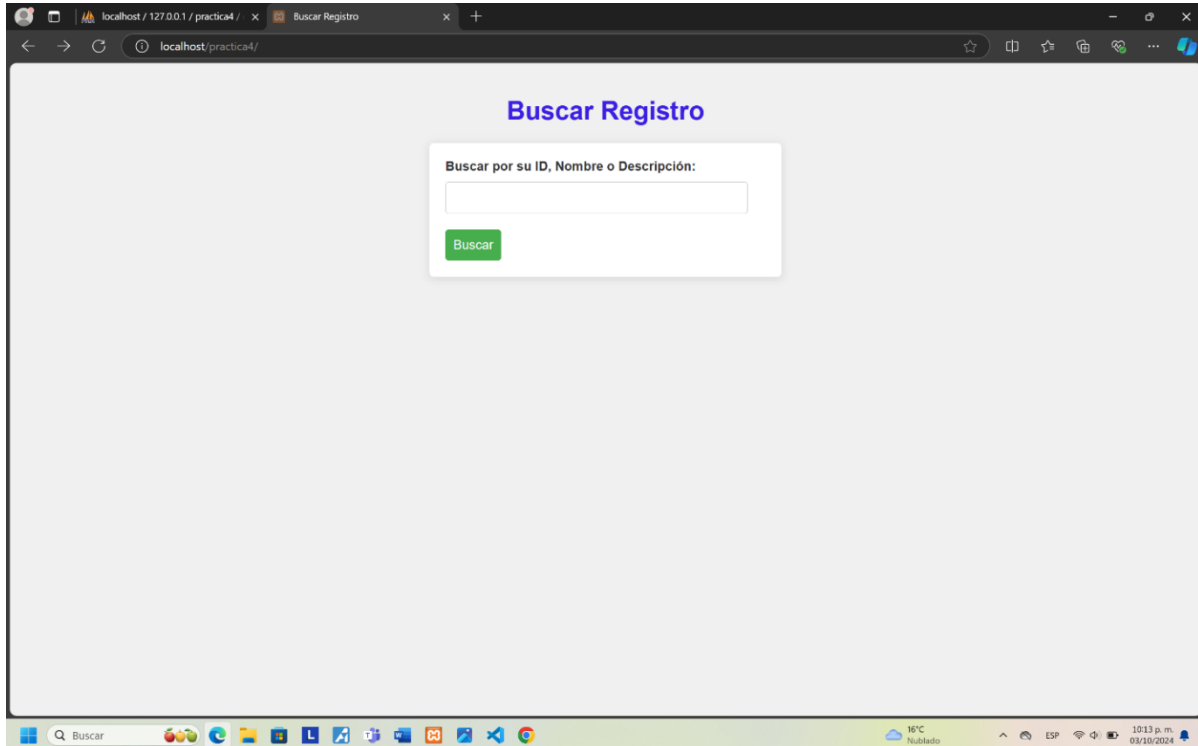
Explicación del archivo de functions.php

Este código PHP permite realizar una búsqueda en una base de datos y mostrar los resultados en una tabla HTML. Utiliza prácticas seguras al preparar consultas SQL y es capaz de manejar situaciones donde no se encuentren resultados o no se proporcione un término de búsqueda.

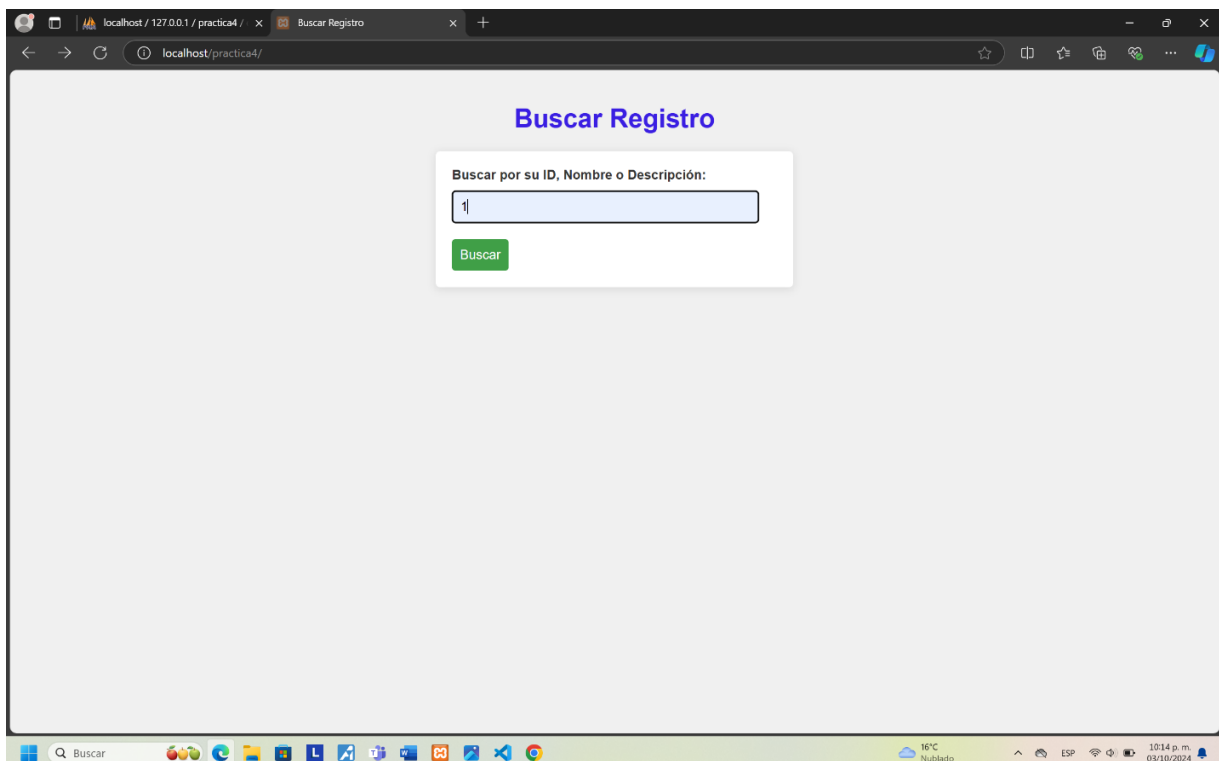
Funcionalidad y pruebas de la aplicación web

Página de búsqueda de un registro

En esta parte se visualiza como esta creado nuestra página web para poder Buscar uno de los tres registros que tenemos en nuestra base de datos.

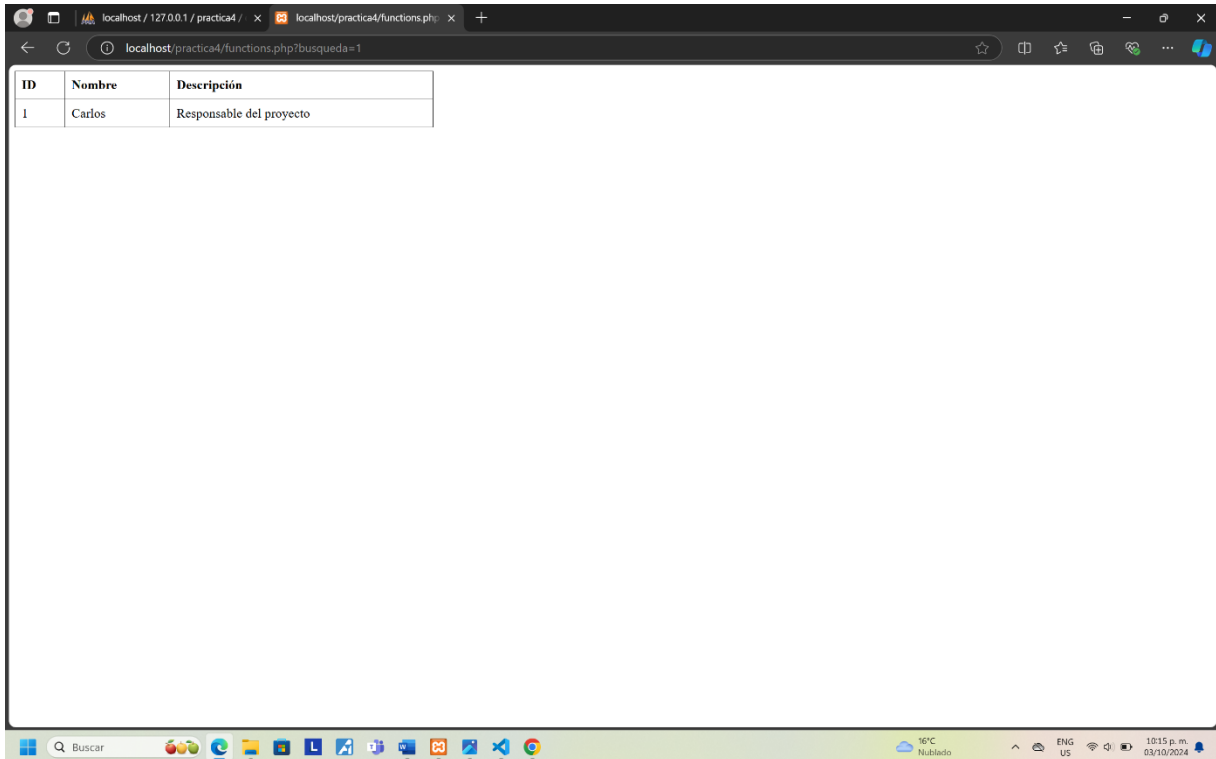


Prueba 1: Para iniciar la búsqueda vamos a buscar el primer **Id=1** para eso colocamos lo siguiente como se muestra en pantalla



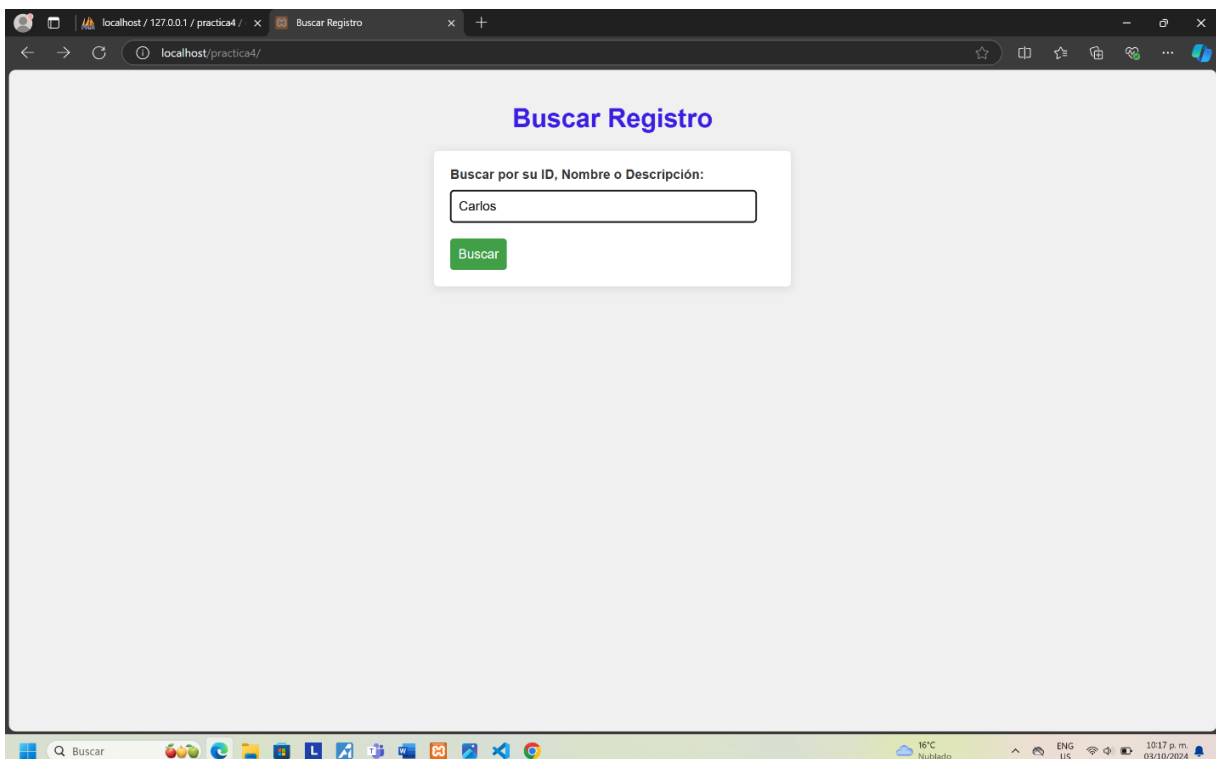


Y por defecto nos debe de mostrar lo siguiente. La siguiente tabla muestra el **ID**, **Nombre y la descripción** de la persona. En este caso, como ID “1” Nombre “Carlos” Descripción “Responsable del proyecto”, esto es necesario para proceder al siguiente paso.



| ID | Nombre | Descripción |
|----|--------|--------------------------|
| 1 | Carlos | Responsable del proyecto |

Prueba 2: Ahora vamos a probar el siguiente método que es buscarlo por su nombre en este caso pondremos **Carlos**. Para esto ay que asegurarnos como estamos buscando su nombre si inicia con una mayúscula o minúscula.



Buscar Registro

Buscar por su ID, Nombre o Descripción:

Buscar



Una vez hecho lo anterior, al proceder, nos debe arrojar los siguientes datos. Tal y como lo podemos visualizar a continuación.

La imagen muestra una interfaz web en un navegador. En la parte superior, hay una barra de direcciones que indica la URL: localhost/practica4/functions.php?busqueda=Carlos. Debajo de la barra, se encuentra una tabla con tres columnas: ID, Nombre y Descripción. La tabla contiene un solo registro con el ID 1, el nombre Carlos y la descripción Responsable del proyecto.

| ID | Nombre | Descripción |
|----|--------|--------------------------|
| 1 | Carlos | Responsable del proyecto |

Prueba 3: Ahora vamos a probar escribiendo tal como está en la descripción de nuestro registro como se muestra en pantalla.

La imagen muestra una interfaz web en un navegador. En la parte superior, hay una barra de direcciones que indica la URL: localhost/practica4/. Debajo de la barra, se encuentra un formulario con el título "Buscar Registro". El formulario tiene un campo de texto con el texto "Responsable del proyecto" y un botón "Buscar".

Buscar Registro

Buscar por su ID, Nombre o Descripción:

Responsable del proyecto

Buscar



Al haber puesto lo que se muestra anteriormente, nos debe arrojar los datos de la persona. Esto quiere decir que este proceso es correcto.

La imagen muestra una interfaz web en un navegador. En la parte superior, hay una barra de direcciones con la URL `localhost/practica4/functions.php?busqueda=Responsable+del+proyecto`. Debajo, se presenta una tabla con tres columnas: ID, Nombre y Descripción. Solo hay una fila de datos.

| ID | Nombre | Descripción |
|----|--------|--------------------------|
| 1 | Carlos | Responsable del proyecto |

Nota: podemos buscar con el ID, Nombre o Descripción de cada persona ósea de los 3 registros que tenemos almacenados en nuestra base de datos en este caso solamente ocupamos el primer registro para hacer las pruebas.

Comprobación en la base de datos.

Para poder la comprobación solamente podemos ver la tabla en nuestra base de datos y corroborar si es correcto con los datos que nos dio

La imagen muestra la interfaz de un cliente de base de datos. En la parte superior, hay una barra de navegación con opciones como Browse, Structure, SQL, Search, Insert, Export, Import, Privileges y Operations. Debajo, se muestra un mensaje de éxito: "Showing rows 0 - 2 (3 total, Query took 0.0005 seconds.)". A continuación, se muestra el código SQL: `SELECT * FROM `datos``. Hay una sección de "Extra options" con botones para Edit, Copy y Delete. En la parte inferior, se muestra una tabla con tres columnas: id, nombre y descripcion. Hay tres filas de datos.

| id | nombre | descripcion |
|----|--------|------------------------------|
| 1 | Carlos | Responsable del proyecto |
| 2 | Ana | Asistente del jefe |
| 3 | Luis | Encargado de la programación |

- Esta aplicación debe ser vulnerable a inyección de código, esto significa que, si el usuario ingresa un valor malicioso en el campo de búsqueda, la aplicación debe mostrar información que no debería ser accesible o permitir realizar acciones que no deberían ser posibles.

```
# Ejemplo en Python con Flask y SQLite (no usar en producción)
from flask import Flask, request
import sqlite3

app = Flask(__name__)

@app.route('/')
def index():
    return '''
    <form action="/search" method="get">
        <input type="text" name="query" placeholder="Buscar">
        <input type="submit" value="Buscar">
    </form>
    '''

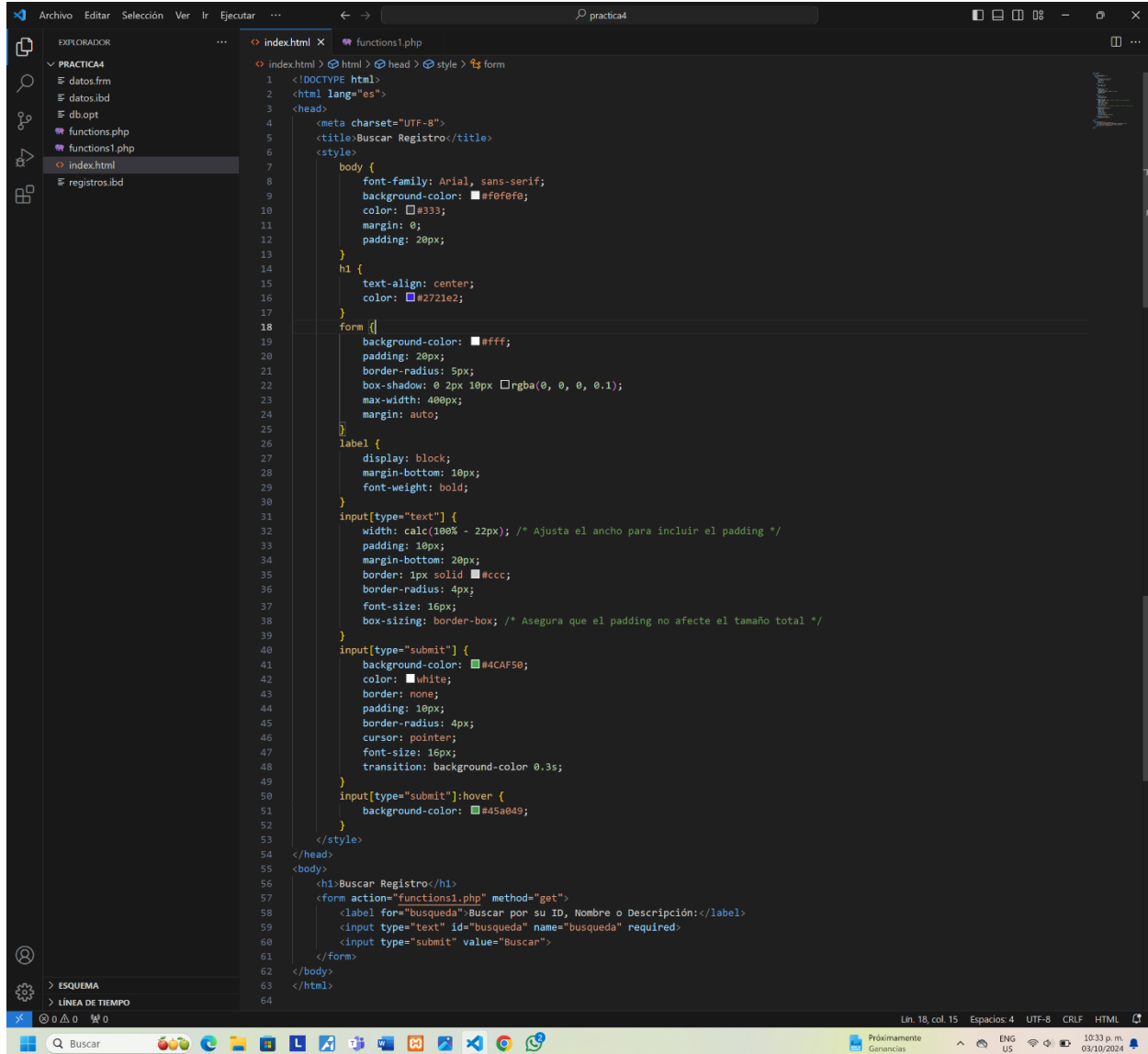
@app.route('/search')
def search():
    query = request.args.get('query')
    conn = sqlite3.connect('database.db')
    c = conn.cursor()
    # Vulnerable a inyección de código
    c.execute(f"SELECT * FROM table WHERE id = '{query}' OR name = '{query}' OR description = '{query}'")
    # Se puede inyectar código SQL en el campo de búsqueda y obtener información no deseada
    # ejemplo:
    # - Buscar por id = 1 OR 1=1 -- Esto devolverá todos los registros de la tabla (lo que no queremos), ya que 1=1 siempre es verdadero y se ignorará el resto de la condición
    # - Buscar por name = ' OR 1=1 --Esto devolverá todos los registros de la tabla (lo que no queremos), ya que 1=1 siempre es verdadero y se ignorará el resto de la condición
    # - Buscar por name = ' OR 1=1; DROP TABLE table; -- Esto eliminará la tabla (lo que no queremos), esto sucede si el usuario ingresa un valor malicioso en el campo de búsqueda y el usuario de la BD tiene permisos para eliminar tablas
    result = c.fetchall()
    conn.close()
    return str(result)

if __name__ == '__main__':
    app.run()
```

3. Realizar pruebas de inyección de código en la aplicación web.

Para este punto ocupamos la misma página solamente creamos otro archivo para que podamos hacer la inyección de código solamente redireccionamos a otra página nueva.

Paso 1: Código utilizado para el index.html

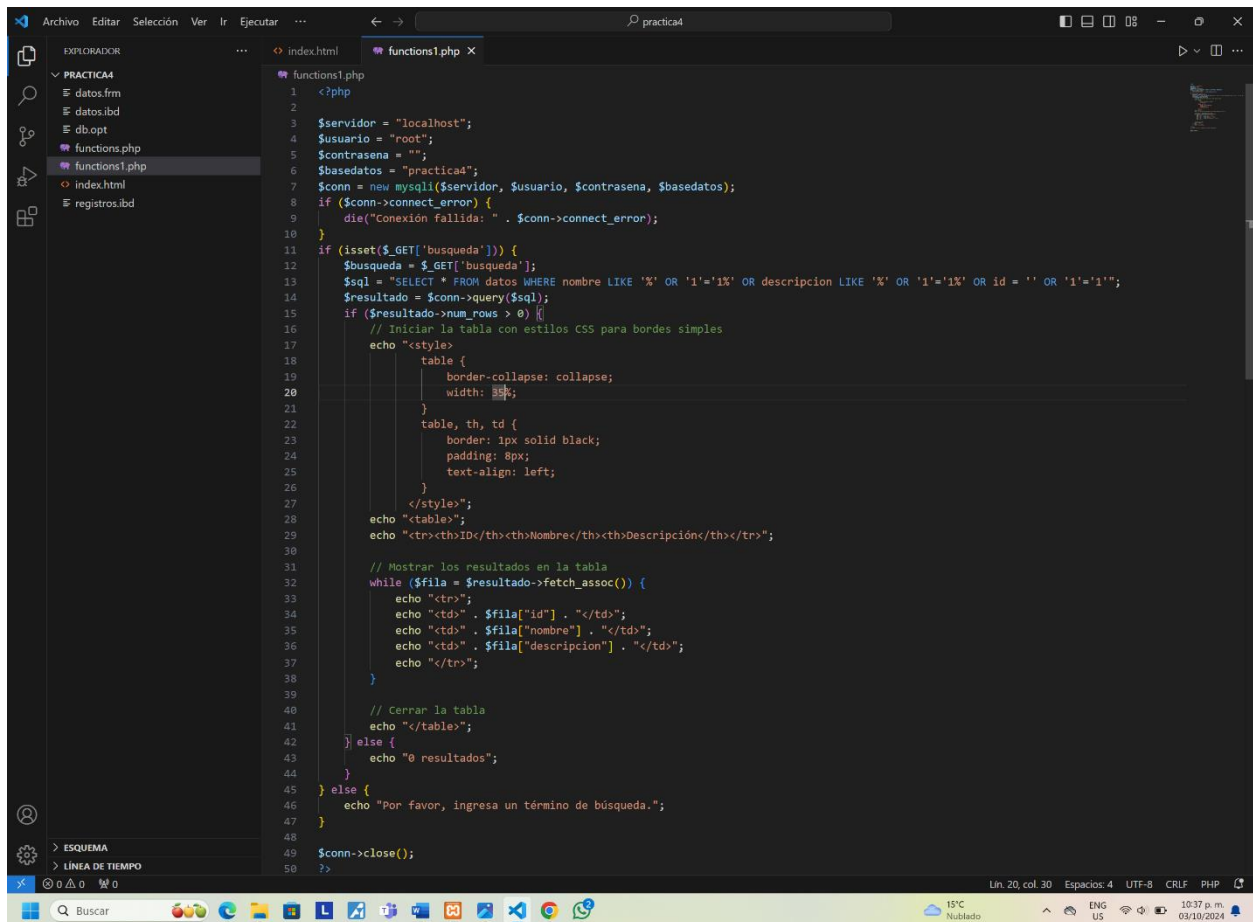


```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4 <meta charset="UTF-8">
5 <title>Buscar Registro</title>
6 <style>
7
8     body {
9         font-family: Arial, sans-serif;
10        background-color: #f0f0f0;
11        color: #333;
12        margin: 0;
13        padding: 20px;
14    }
15    h1 {
16        text-align: center;
17        color: #272e2e;
18    }
19    form {
20        background-color: #fff;
21        padding: 20px;
22        border-radius: 5px;
23        box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
24        max-width: 400px;
25        margin: auto;
26    }
27    label {
28        display: block;
29        margin-bottom: 10px;
30        font-weight: bold;
31    }
32    input[type="text"] {
33        width: calc(100% - 22px); /* Ajusta el ancho para incluir el padding */
34        padding: 10px;
35        margin-bottom: 20px;
36        border: 1px solid #ccc;
37        border-radius: 4px;
38        font-size: 16px;
39        box-sizing: border-box; /* Asegura que el padding no afecte el tamaño total */
40    }
41    input[type="submit"] {
42        background-color: #4CAF50;
43        color: white;
44        border: none;
45        padding: 10px;
46        border-radius: 4px;
47        cursor: pointer;
48        font-size: 16px;
49        transition: background-color 0.3s;
50    }
51    input[type="submit"]:hover {
52        background-color: #45a049;
53    }
54 </style>
55 </head>
56 <body>
57 <h1>Buscar Registro</h1>
58 <form action="funcions1.php" method="get">
59 <label for="busqueda">Buscar por su ID, Nombre o Descripción:</label>
60 <input type="text" id="busqueda" name="busqueda" required>
61 <input type="submit" value="Buscar">
62 </form>
63 </body>
64 </html>
```

Explicación del código utilizado en index.html

Este código crea una página web simple que permite a los usuarios buscar registros mediante un formulario. Está bien estructurado y estilizado para ser atractivo visualmente, y es funcional al enviar las búsquedas a un archivo PHP para su procesamiento.

Paso 2. Código utilizado para la función de inyección (functions1.php)



```
1 <?php
2
3 $servidor = "localhost";
4 $usuario = "root";
5 $contrasena = "";
6 $basedatos = "practicad";
7 $conn = new mysqli($servidor, $usuario, $contrasena, $basedatos);
8 if ($conn->connect_error) {
9     die("Conexión fallida: " . $conn->connect_error);
10 }
11 if (isset($_GET['busqueda'])) {
12     $busqueda = $_GET['busqueda'];
13     $sql = "SELECT * FROM datos WHERE nombre LIKE '%" . $busqueda . "%' OR '1'='1%' OR descripcion LIKE '%" . $busqueda . "%' OR '1'='1%' OR id = '' OR '1'='1%'";
14     $resultado = $conn->query($sql);
15     if ($resultado->num_rows > 0) {
16         // Iniciar la tabla con estilos CSS para bordes simples
17         echo "<style>";
18         echo "table {";
19             echo "border-collapse: collapse;";
20             echo "width: 80%;";
21         echo "table, th, td {";
22             echo "border: 1px solid black;";
23             echo "padding: 8px;";
24             echo "text-align: left;";
25         echo "}</style>";
26         echo "<table>";
27         echo "<tr><th>ID</th><th>Nombre</th><th>Descripción</th></tr>";
28
29         // Mostrar los resultados en la tabla
30         while ($fila = $resultado->fetch_assoc()) {
31             echo "<tr>";
32             echo "<td>" . $fila["id"] . "</td>";
33             echo "<td>" . $fila["nombre"] . "</td>";
34             echo "<td>" . $fila["descripcion"] . "</td>";
35             echo "</tr>";
36         }
37
38         // Cerrar la tabla
39         echo "</table>";
40     } else {
41         echo "0 resultados";
42     }
43 } else {
44     echo "Por favor, ingresa un término de búsqueda.";
45 }
46 $conn->close();
47 ?>
```

Explicación del archivo de functions1.php

Este código permite realizar búsquedas en una tabla de base de datos, pero tiene un grave problema de seguridad por la forma en que se construye la consulta SQL. Para corregir esto, se deben usar consultas preparadas en lugar de concatenar directamente la entrada del usuario en la consulta. Esto evitaría la inyección SQL y haría que la aplicación sea más segura.



Funcionalidad y pruebas de la aplicación web

Prueba 1: Para iniciar la búsqueda vamos a buscar el primer **Id=1** para eso colocamos lo siguiente como se muestra en pantalla

La imagen muestra una ventana de navegador web con la URL `localhost/practica4/`. El título de la página es "Buscar Registro". Hay un campo de texto con el placeholder "Buscar por su ID, Nombre o Descripción:" y el valor "1" ingresado. Debajo del campo hay un botón verde que dice "Buscar".

Y nos debe mostrar todas las tablas que tengamos en nuestra base de datos ya que así estemos haciendo la inyección.

La imagen muestra la misma ventana de navegador web, pero ahora la URL es `localhost/practica4/functions1.php?busqueda=1`. Se muestra una tabla con tres columnas: ID, Nombre y Descripción. La tabla contiene tres filas de datos.

| ID | Nombre | Descripción |
|----|--------|------------------------------|
| 1 | Carlos | Responsable del proyecto |
| 2 | Ana | Asistente del jefe |
| 3 | Luis | Encargado de la programación |



Prueba 2: Ahora vamos a probar el siguiente método que es buscarlo por su nombre en este caso pondremos el nombre del segundo registro que tenemos en nuestra base de datos que es **Ana**. Para esto ay que asegurarnos como lo estamos buscando si su nombre inicia con mayúscula o minúscula.

Buscar Registro

Buscar por su ID, Nombre o Descripción:

Ana

Buscar

Y de igual forma hace la inyección y nos muestra los registros que tenemos en nuestra base de datos.

| ID | Nombre | Descripción |
|----|--------|------------------------------|
| 1 | Carlos | Responsable del proyecto |
| 2 | Ana | Asistente del jefe |
| 3 | Luis | Encargado de la programación |



Prueba 3: Ahora vamos a probar escribiendo tal como está en la descripción de uno de los registros que tengamos, así como se muestra en pantalla.

La interfaz muestra un formulario de búsqueda con el título "Buscar Registro". Debajo del título, hay un campo de texto con el placeholder "Buscar por su ID, Nombre o Descripción:". El campo contiene el texto "Encargado de la programación". Debajo del campo, hay un botón verde con el texto "Buscar".

Este proceso nos debe mostrar los datos que tiene nuestra base de datos haciendo la inyección en ella. Tal y como se presenta a continuación.

La interfaz muestra una tabla con los resultados de la búsqueda. La URL de la página es "localhost/practica4/functions1.php?busqueda=Encargado+de+la+programación". La tabla tiene tres columnas: ID, Nombre y Descripción.

| ID | Nombre | Descripción |
|----|--------|------------------------------|
| 1 | Carlos | Responsable del proyecto |
| 2 | Ana | Asistente del jefe |
| 3 | Luis | Encargado de la programación |

4. Documentar los resultados obtenidos y las posibles acciones que se pueden realizar para prevenir este tipo de vulnerabilidades.

DOCUMENTACIÓN DE RESULTADOS OBTENIDOS

1. *Descripción de la Vulnerabilidad:*

- Durante las pruebas realizadas, se identificó que la aplicación web es vulnerable a ataques de inyección de código SQL. La inyección SQL es un tipo de ataque en el cual el atacante puede manipular las consultas SQL que se ejecutan en el servidor al inyectar comandos maliciosos a través de las entradas de la aplicación. Esto suele ocurrir cuando las consultas se construyen concatenando directamente los valores ingresados por el usuario sin la debida validación o protección.
- Esta vulnerabilidad es crítica porque permite a un atacante obtener acceso a información confidencial, como datos de usuarios, o incluso realizar modificaciones no autorizadas, tales como la alteración o eliminación de registros en la base de datos.

2. *Pruebas Realizadas:*

- Se realizaron pruebas inyectando entradas maliciosas en el campo de búsqueda de la aplicación. Las pruebas consistieron en utilizar operadores lógicos (OR 1=1 --) y otras técnicas que permiten manipular la lógica de las consultas SQL. Por ejemplo, se intentó una búsqueda con `Id=1 OR 1=1 --`, lo cual tiene como propósito modificar la lógica de la consulta para que siempre sea verdadera y devolver todos los registros de la base de datos.
- También se intentó inyectar comandos para borrar tablas (DROP TABLE). Esto demostró que, si el usuario que ejecuta la consulta tiene privilegios suficientes, el atacante podría eliminar toda la información almacenada.

3. *Resultados:*

- Los resultados de las pruebas demostraron que la aplicación no cuenta con los controles de seguridad necesarios para evitar la inyección SQL. Al ingresar valores maliciosos, la aplicación devolvió todos los registros de la base de datos, sin restricciones, y en algunos casos permitiría realizar acciones destructivas si el atacante tiene permisos elevados.
- La falta de protección en el manejo de las consultas SQL representa una amenaza significativa, ya que pone en riesgo la confidencialidad, integridad y disponibilidad de los datos.

POSIBLES ACCIONES PARA PREVENIR VULNERABILIDADES

1. *Uso de Consultas Preparadas (Prepared Statements):*

- **Descripción:** Una consulta preparada es una función del lenguaje SQL que permite definir la estructura de la consulta de antemano, antes de que los datos del usuario sean añadidos. En lugar de insertar directamente los valores proporcionados por el usuario, estos se asignan a parámetros predefinidos de la consulta.
- **Argumentación:** El uso de consultas preparadas garantiza que los datos ingresados por el usuario no puedan alterar la lógica de la consulta, ya que se tratan siempre como valores en lugar de comandos SQL. Esto evita la inyección de código porque cualquier entrada proporcionada se procesa como un simple dato. Es una práctica recomendada por los principales frameworks y es compatible con la mayoría de las bases de datos modernas.

2. *Validación de Entradas del Usuario:*

- **Descripción:** Implementar validación rigurosa de los datos ingresados por el usuario para asegurarse de que solo se permitan entradas seguras. Esto incluye comprobar el tipo de datos (por ejemplo, asegurarse de que el ID sea siempre numérico) y restringir el tamaño y los caracteres permitidos.
- **Argumentación:** Al restringir los tipos de entradas permitidos, se reduce la superficie de ataque disponible para un atacante. Por ejemplo, al asegurarse de que los campos numéricos solo contengan números, se elimina la posibilidad de inyectar comandos SQL. Además, se puede evitar el uso de caracteres especiales como ' o --, que suelen ser utilizados en los ataques de inyección.

3. *Escapar Caracteres Especiales:*

- **Descripción:** Escapar todos los caracteres especiales que podrían alterar la sintaxis de una consulta SQL, como comillas simples ('), comillas dobles ("), y el operador de comentario (--).
- **Argumentación:** Aunque no es una solución tan robusta como las consultas preparadas, escapar caracteres especiales puede ser una defensa adicional que ayude a evitar que las entradas maliciosas afecten la lógica de la consulta. Sin embargo, esta técnica debe ser utilizada en combinación con otras medidas de seguridad, ya que puede no ser suficiente por sí sola.

4. *Manejo de Errores:*

- **Descripción:** Implementar un sistema de manejo de errores adecuado para la aplicación, que no exponga detalles técnicos al usuario en caso de un fallo.
- **Argumentación:** Mostrar mensajes de error detallados que incluyen información sobre las consultas SQL o la estructura de la base de datos puede proporcionar información valiosa a un atacante. Utilizando mensajes de error genéricos y registrando los detalles técnicos para el equipo de desarrollo, se dificulta que los atacantes comprendan cómo manipular la aplicación de manera maliciosa.

5. *Uso de Módulos o Bibliotecas de Seguridad:*

- **Descripción:** Utilizar frameworks o bibliotecas que ofrezcan funciones de protección contra inyecciones SQL. Estos frameworks suelen incluir mecanismos automáticos para evitar que la entrada del usuario se utilice de forma insegura en las consultas.
- **Argumentación:** Los frameworks modernos como Django (Python), Laravel (PHP), y otros, ya implementan mecanismos de protección contra las inyecciones SQL al manejar las consultas a la base de datos. Al aprovechar estas herramientas, se disminuye el riesgo de vulnerabilidades debido a errores de desarrollo.

6. *Configuración de la Base de Datos:*

- **Descripción:** Limitar los privilegios del usuario de la base de datos utilizado por la aplicación. Este usuario debe tener únicamente los permisos necesarios para ejecutar las operaciones que requiere la aplicación.
- **Argumentación:** En caso de que se explote una vulnerabilidad, limitar los permisos puede reducir significativamente el impacto del ataque. Por ejemplo, si el usuario de la base de datos no tiene permisos para eliminar tablas, un ataque que intente ejecutar DROP TABLE fallará. Es buena práctica otorgar el menor nivel de privilegios posible para minimizar los riesgos.

La prevención de la inyección SQL requiere una combinación de buenas prácticas de programación y una configuración adecuada de la base de datos y el entorno de la aplicación. Las consultas preparadas ofrecen la mejor defensa técnica, ya que evitan por completo que las entradas del usuario se utilicen de forma insegura en consultas SQL. Al implementar una validación adecuada de los datos del usuario, escapar caracteres especiales y utilizar bibliotecas de seguridad, es posible minimizar el riesgo de inyecciones SQL. Además, limitar los privilegios del usuario de la base de datos y manejar los errores de forma segura añade capas adicionales de protección, asegurando que la aplicación sea más robusta y resistente frente a los ataques.

5. Investiga y describe los siguientes conceptos:

- Inyección de SQL
- Inyección SQL a ciegas
- Inyección SQL basada en errores
- Inyección SQL basada en el tiempo
- Inyección SQL en procedimientos almacenados
- Inyección SQL y ORM
- Herramientas para detectar y prevenir la inyección SQL

Inyección de SQL: La inyección de SQL (SQL Injection, o SQLi) es una vulnerabilidad de seguridad en aplicaciones web que ocurre cuando un atacante inserta o "inyecta" código SQL malicioso en una consulta SQL, aprovechando una mala validación o sanitización de los datos ingresados por el usuario. Esto permite al atacante manipular las consultas que la aplicación envía a la base de datos, potencialmente logrando acceso no autorizado a información sensible, modificando o eliminando datos, o incluso tomando el control completo del servidor de base de datos.

Inyección SQL a ciegas: La inyección SQL a ciegas ocurre cuando una aplicación es vulnerable a inyección SQL, pero no devuelve mensajes de error que permitan al atacante obtener información directa. En lugar de basarse en errores visibles, el atacante envía consultas SQL maliciosas y observa indirectamente el comportamiento del sistema (por ejemplo, tiempos de respuesta o cambios en la página) para deducir si la inyección fue exitosa. Es un ataque más difícil de realizar, pero igual de peligroso.

Inyección SQL basada en errores: La inyección SQL basada en errores aprovecha los mensajes de error generados por la base de datos o la aplicación cuando una consulta SQL falla. Estos errores pueden proporcionar detalles sobre la estructura de la base de datos o las consultas SQL subyacentes, lo que ayuda al atacante a ajustar sus inyecciones para obtener información confidencial o realizar otras acciones maliciosas. Es una forma común de inyección cuando el sistema no maneja adecuadamente los errores.

Inyección SQL basada en el tiempo: En la inyección SQL basada en el tiempo, el atacante introduce consultas SQL que generan retrasos o pausas en la respuesta del servidor para deducir si una consulta es exitosa o no. Este método se utiliza cuando la aplicación no proporciona retroalimentación visible, pero el atacante puede medir el tiempo que tarda en responder el servidor. Dependiendo de si se cumple una condición en la base de datos, el servidor tardará más o menos en responder.

Inyección SQL en procedimientos almacenados: Los procedimientos almacenados son funciones predefinidas en las bases de datos que pueden ser vulnerables a inyección SQL si no se manejan adecuadamente los parámetros que se les pasan. Aunque los procedimientos almacenados suelen mejorar el rendimiento y la seguridad, pueden ser manipulados mediante inyecciones si se permite la inserción de código SQL sin validación. Esto puede llevar a la ejecución de consultas maliciosas dentro de los propios procedimientos.

Inyección SQL y ORM: El ORM (Object-Relational Mapping) es una técnica que permite a los desarrolladores trabajar con bases de datos utilizando objetos en lugar de consultas SQL directas. Aunque el uso de un ORM puede reducir el riesgo de inyección SQL, no lo elimina completamente. Las inyecciones pueden ocurrir si los desarrolladores no utilizan correctamente las características de seguridad del ORM o si desactivan las protecciones automáticas, como la sanitización de entradas.

La inyección SQL y ORM se refiere a cómo las vulnerabilidades de inyección SQL pueden estar presentes incluso en aplicaciones que utilizan un Object-Relational Mapping (ORM), una técnica que mapea objetos en el código a tablas en una base de datos, permitiendo a los desarrolladores interactuar con la base de datos usando estructuras de datos de un lenguaje de programación en lugar de escribir directamente consultas SQL.

Herramientas para detectar y prevenir la inyección SQL

Existen varias herramientas que ayudan a detectar y prevenir la inyección SQL. Algunas de las más comunes son:

- **Sqlmap:** Una herramienta de código abierto para detectar y explotar vulnerabilidades de inyección SQL en aplicaciones web.
- **OWASP ZAP:** Un escáner de seguridad para aplicaciones web que puede detectar inyecciones SQL.
- **WAF (Web Application Firewall):** Un firewall especializado en proteger aplicaciones web de amenazas, incluyendo la inyección SQL.
- **Prepared Statements:** Una técnica de programación que previene la inyección SQL al separar el código SQL de los datos ingresados por el usuario, impidiendo que el atacante inserte código malicioso.
- **Snyk y SonarQube:** Herramientas que analizan el código para detectar vulnerabilidades de seguridad, incluida la inyección SQL.



Conclusión

Realizar esta práctica es muy importante porque nos permite entender de manera clara cómo funciona una vulnerabilidad de inyección SQL y cómo los atacantes pueden aprovecharse de ella. Al hacer las pruebas y ver los resultados, podemos darnos cuenta de lo fácil que es acceder a información que no debería ser visible o incluso modificar y eliminar datos sensibles. Además, esta práctica nos ayuda a darnos cuenta de la importancia de implementar medidas de seguridad en nuestras aplicaciones, como validar adecuadamente las entradas del usuario y utilizar consultas seguras. Si no tomamos estas precauciones, dejamos nuestras aplicaciones y bases de datos expuestas a ataques que pueden causar daños graves.

Concluyendo esto, aprender sobre inyección SQL y cómo prevenirla es fundamental para crear aplicaciones web más seguras y proteger tanto los datos como los usuarios que confían en nuestros sistemas. Es así como se ha terminado esta práctica y como equipo de trabajo, realizarlo fue de gran importancia para comprender más sobre el tema.



Bibliografías

- <https://www.avast.com/es-es/c-sql-injection>
- <https://keepcoding.io/blog/tipos-de-inyeccion-sql/>
- <https://latam.kaspersky.com/resource-center/definitions/sql-injection>
- <https://ricardogeek.com/ataques-de-inyeccion-sql-ciegos-basados-en-el-tiempo/>
- <https://www.cloudflare.com/es-es/learning/security/threats/how-to-prevent-sql-injection/>
- <https://solutioncenter.apexsql.com/es/como-asegurar-procedimientos-almacenados-contrainyecciones-sql/>
- <https://blog.hackmetrix.com/sql-injection/#:~:text=El%20ORM%20es%20un%20modelo,f%C3%A1cilmente%20y%20cuando%20sea%20requerido.>