



TECNOLÓGICO
NACIONAL DE MÉXICO®



TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

PRACTICA 4 - INYECCIÓN SQL

Yael de Jesús Santiago Ortiz

Javier Noé Cruz España

Kevin Sanches Hernández

**PROFESOR: ING. EDWARD
OSORIO SALINAS**

SEGURIDAD Y VIRTUALIZACIÓN

1U

7° US

**ING SISTEMAS
COMPUTACIONALES**

Contenido

OBJETIVO.....	3
INSTRUCCIONES.....	3
1. Crear una base de datos con una tabla que contenga al menos 3 registros. ...	3
2. Crear una aplicación web que permita buscar un registro por su id, nombre o descripción.....	4
3. Realizar pruebas de inyección de código en la aplicación web.....	10
INVESTIGACIÓN	11
1. Inyección de SQL.....	11
2. Blind SQL Injection	11
3. SQL Injection basada en errores.....	12
4. SQL Injection basada en tiempo	12
5. SQL Injection en procedimientos almacenados	13
6. SQL Injection en ORM	13
7. Herramientas para detectar y prevenir SQL Injection.....	13
CONCLUSIÓN.....	14
BIBLIOGRAFÍA.....	15

OBJETIVO

Crear una base de datos vulnerable a inyección de código y demostrar como se puede explotar esta vulnerabilidad.

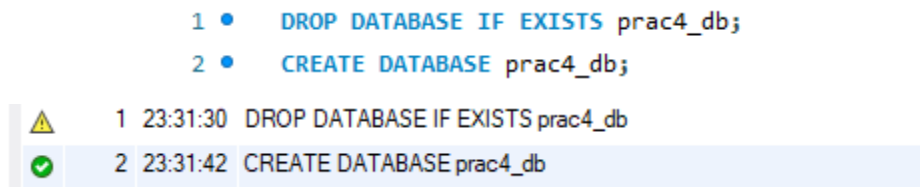
INSTRUCCIONES

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

El script SQL crea una base de datos prac4_db, una tabla llamada registros, y la llena con tres registros.

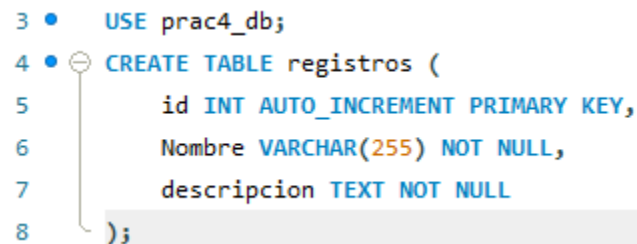
```
DROP DATABASE IF EXISTS prac4_db;
```

```
CREATE DATABASE prac4_db;
```



```
USE prac4_db;
```

```
CREATE TABLE registros (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    Nombre VARCHAR(255) NOT NULL,  
    descripcion TEXT NOT NULL  
);
```



Ahora insertamos los datos para verificar la tabla

```
INSERT INTO registros (Nombre , descripcion) VALUES
```

```
("Yael DE JESUS SANTIAGO ORTIZ", "ESTUDIANTE DEL 7°  
SEMESTRE"),
```





```
("JAVIER NOE CRUZ ESPAÑA", "ESTUDIANTE DEL 7° SEMESTRE"),
```

```
("KEVIN SANCHES HERNÁNDEZ", "ESTUDIANTE DEL 7º SEMESTRE");
```

```
9 • INSERT INTO registros (Nombre , descripcion) VALUES
10 ("Yael de Jesus Santiago Ortiz", "ESTUDIANTE DEL 7º SEMESTRE"),
11 ("Javier Noe Cruz España", "ESTUDIANTE DEL 7º SEMESTRE"),
12 ("KEVIN SANCHES HERNÁNDEZ", "ESTUDIANTE DEL 7º SEMESTRE");
```

1 00:18:43 INSERT INTO registros (Nombre , descripcion) VALUES ("Yael de Jesus Santiago Ortiz", "ESTUDIANTE DEL 7º SEMESTRE") 3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0

```
14 • SELECT * FROM registros;
```

Result Grid			
Filter Rows: <input type="text"/>			
Edit:    			
	id	Nombre	descripcion
▶	1	Yael de Jesus Santiago Ortiz	ESTUDIANTE DEL 7º SEMESTRE
	2	Javier Noe Cruz España	ESTUDIANTE DEL 7º SEMESTRE
	3	KEVIN SANCHES HERNÁNDEZ	ESTUDIANTE DEL 7º SEMESTRE
*	NULL	NULL	NULL

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

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.

Conectamos la base de datos con la aplicación web.

```
<center>
```

```
<?php
```

```
// Configuración de conexión a la base de datos
```

```
$host = 'localhost'; // Cambia si tu servidor es diferente
```

```
$user = 'root'; // Cambia si tienes otro usuario
```

```
$password = ''; // Cambia si tienes una contraseña
```

```
$dbname = 'prac4_db';
```

```
try {
```

```
    // Crear una conexión PDO
```

```

        $pdo = new
PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $user,
$password);

        $pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
            if (!empty($_POST['buscar'])) {
                $buscar = htmlspecialchars($_POST['buscar']); //
Sanitizar el input
                // Consulta preparada para evitar inyección SQL
                $stmt = $pdo->prepare("SELECT * FROM registros
WHERE id = :id OR Nombre LIKE :buscar OR descripcion LIKE
:buscar");

                // Vinculamos los valores a la consulta preparada
                $stmt->execute([
                    'id' => is_numeric($buscar) ? $buscar : -1,
// Usamos el ID si es un número; de lo contrario, asignamos
un valor imposible
                    'buscar' => "%$buscar%"
                ]);
                $resultados = $stmt->fetchAll(PDO::FETCH_ASSOC);
                if (count($resultados) > 0) {
                    echo "<h1>Resultados de la búsqueda</h1>";
                    echo "<table border='1'>";
                    echo
"<tr><th>ID</th><th>Nombre</th><th>Descripción</th></tr>";
                    foreach ($resultados as $fila) {
                        echo "<tr>";
                        echo "<td>" . $fila['id'] . "</td>";
                        echo
                            "<td>"
                                .
htmlspecialchars($fila['Nombre']) . "</td>";

```

```

        echo "
        "
        htmlspecialchars($fila['descripcion']) . "</td>";
        echo "</tr>";
    }
    echo "</table>";
} else {
    echo "<p>No se encontraron resultados para:
<strong>" . $buscar . "</strong></p>";
}
} else {
    echo "<p>Por favor, ingresa un término para
buscar.</p>";
}
}
} catch (PDOException $e) {
    // Manejo de errores de conexión
    echo "Error de conexión a la base de datos: " . $e-
>getMessage();
}
?>

```

Después la interfaz y comprobamos su funcionamiento.

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>FORMULARIO 1</title>
<style>
    .container {
        display: grid;
        gap: 5px;
        border: 2px solid rgb(0, 0, 0);
        padding: 50px;
        width: 30%;
        margin: auto;
    }
    .container div {
        width: 100%;
    }
    .container label {
        display: flex;
        color: black;
    }
    .container label span {
        color: red;
        font-weight: normal;
        margin-left: 5px;
    }
    .container input[type="text"],
    .container select {
        width: calc(100% - 20px);
        padding: 10px;
        box-sizing: border-box;
    }
    table {
```

```
        width: 100%;
    }
    table td {
        text-align: center;
    }
    form {
        text-align: center;
    }
</style>
</head>
<body>
    <div class="container">
        <center><h1>Beneficiario</h1></center>
        <!-- Aquí añadimos el action -->
        <form method="POST" action="buscar.php">
            <div>
                <label for="BUSCAR">Ingresar Datos: <span>*</span></label>
                <input type="text" id="BUSCAR" name="buscar" placeholder="Ingrese
su Id, Nombre o Descripción">
            </div>
            <br/>
            <input type="submit" value="BUSCAR">
        </form>
        <br/>
    </div>
</body>
</html>
```


Beneficiario

Ingresar Datos: *

ID	Nombre	Descripción
1	Yael de Jesus Santiago Ortiz	Estudiante del 7° semestre

Atrás

ID	Nombre	Descripción
2	JAVIER NOE CRUZ ESPAÑA	ESTUDIANTE DEL 7º SEMESTRE

[Atrás](#)

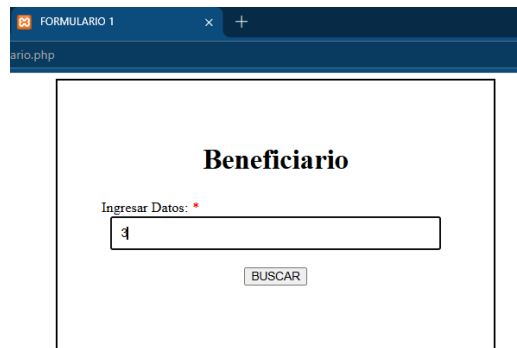
ID	Nombre	Descripción
1	Yael de Jesus Santiago Ortiz	Estudiante del 7º semestre
2	Javier Noe Cruz España	Estudiante del 7º semestre
3	Kevin Sanches Hernández	Estudiante del 7º semestre

[Atrás](#)

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

Realizamos pruebas con inyección que en lugar de mostrar correctamente a la búsqueda de id mostrando solo uno como en el ejemplo 1 mostrara todos los resultados posibles sin importar la id como se muestra en el ejemplo 2:

Ejemplo 1:



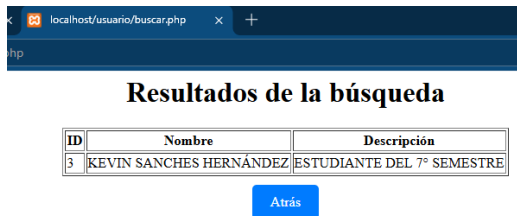
FORMULARIO 1

ario.php

Beneficiario

Ingresar Datos: *

BUSCAR



localhost/usuario/buscar.php

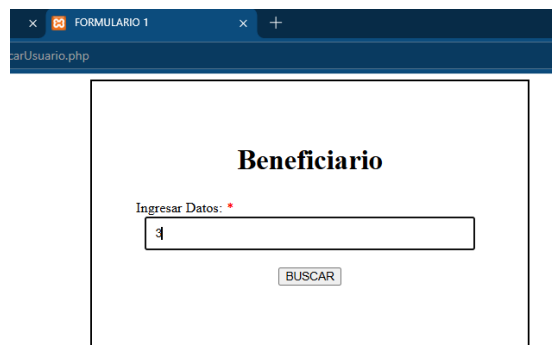
hp

Resultados de la búsqueda

ID	Nombre	Descripción
3	KEVIN SANCHES HERNÁNDEZ	ESTUDIANTE DEL 7º SEMESTRE

Atrás

Ejemplo 2:



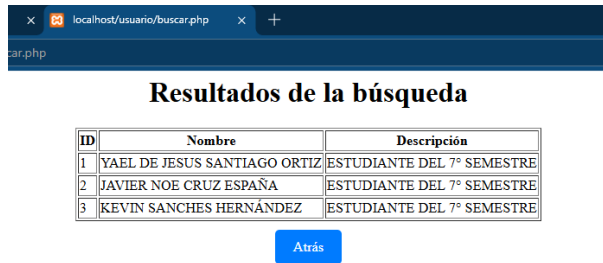
FORMULARIO 1

carUsuario.php

Beneficiario

Ingresar Datos: *

BUSCAR



The screenshot shows a web browser window with the address bar displaying 'localhost/usuario/buscar.php'. The page title is 'Resultados de la búsqueda'. Below the title is a table with three columns: 'ID', 'Nombre', and 'Descripción'. The table contains three rows of data. Below the table is a blue button labeled 'Atrás'.

ID	Nombre	Descripción
1	Yael de Jesus Santiago Ortiz	Estudiante del 7º semestre
2	Javier Noe Cruz España	Estudiante del 7º semestre
3	Kevin Sanches Hernández	Estudiante del 7º semestre

Atrás

INVESTIGACIÓN

1. Inyección de SQL

La inyección de SQL es una vulnerabilidad que permite a un atacante manipular las consultas SQL que una aplicación realiza a una base de datos. Esto se logra insertando código SQL malicioso en los campos de entrada de la aplicación, lo que puede llevar a accesos no autorizados, robo de datos o la modificación/eliminación de información.

Características:

- Ocurre cuando las entradas no son validadas ni parametrizadas.
- Permite realizar operaciones no autorizadas en la base de datos.

Ejemplo de consulta vulnerable:

```
SELECT * FROM users WHERE username = 'user' AND password = 'pass';
```

Si un atacante envía `user' OR '1'='1`, la consulta se manipula para devolver todos los registros.

2. Blind SQL Injection

En una inyección SQL "a ciegas", el atacante no recibe mensajes de error detallados, pero puede inferir información observando los cambios en el comportamiento de la aplicación o el tiempo de respuesta.

Características:

- Se explotan las respuestas binarias (verdadero/falso) de la base de datos.

Métodos comunes:

- Basada en contenido: Se verifica si ciertos resultados se muestran o no.
- Basada en tiempo: Se usan comandos que retrasan la respuesta para determinar si una condición es verdadera.

Ejemplo:

```
SELECT * FROM users WHERE id = 1 AND IF(1=1, SLEEP(5), 0);
```

3. SQL Injection basada en errores

Este tipo de ataque explota los mensajes de error que genera la base de datos. Los errores pueden revelar información sensible como nombres de tablas, columnas o detalles del sistema.

Características:

- El atacante fuerza la aplicación a generar errores detallados.

Ejemplo:

```
SELECT * FROM users WHERE id = 1' AND 1=CONVERT(int, (SELECT @@version));
```

Esto podría revelar la versión del servidor SQL.

4. SQL Injection basada en tiempo

Este ataque se utiliza cuando no se reciben respuestas directas ni errores. El atacante utiliza comandos que introducen retardos en la respuesta para inferir si una condición es verdadera.

Características:

- Se basa en funciones como SLEEP o WAITFOR DELAY.

Ejemplo:

```
SELECT * FROM users WHERE id = 1 AND IF(1=1, SLEEP(5), 0);
```

Si la consulta tarda más tiempo en responder, la condición es verdadera.

5. SQL Injection en procedimientos almacenados

Los procedimientos almacenados son fragmentos de SQL precompilados que pueden ser explotados si aceptan entradas no validadas. Este ataque puede ser más difícil de detectar si las consultas se generan dinámicamente dentro del procedimiento.

Características:

- A menudo ocurre cuando los procedimientos concatenan entradas directamente.

Ejemplo vulnerable:

```
CREATE PROCEDURE GetUserData (@userId NVARCHAR(MAX))
```

```
AS
```

```
EXEC('SELECT * FROM users WHERE id = ' + @userId);
```

6. SQL Injection en ORM

Los ORM (Object-Relational Mapping) son herramientas que abstraen las consultas SQL. Sin embargo, algunos ORM pueden ser vulnerables si se construyen consultas dinámicas a partir de entradas sin validar.

Características:

- Los ORM como Hibernate o Sequelize pueden ser explotados si se usan métodos inseguros.

Ejemplo vulnerable:

```
session.query(User).filter(f"id = {user_input}").all()
```

Si user_input no se valida, el ataque puede inyectar SQL malicioso.

7. Herramientas para detectar y prevenir SQL Injection

Detección

SQLMap: Herramienta automatizada para detectar y explotar vulnerabilidades de SQL Injection.

Burp Suite: Ayuda a identificar inyecciones SQL al analizar tráfico HTTP/HTTPS.

OWASP ZAP: Similar a Burp Suite, útil para pruebas de seguridad automatizadas.

Nmap NSE Scripts: Incluye scripts que pueden detectar aplicaciones vulnerables.

Prevención

Parametrización de consultas: Usar consultas preparadas o Prepared Statements evita la concatenación directa de datos de entrada.

Validación de entradas: Asegúrate de que los datos sean válidos antes de usarlos en consultas.

Escapes seguros: Escapar caracteres especiales según el motor de base de datos.

Cortafuegos de aplicaciones web (WAF): Protege contra ataques al monitorear y filtrar solicitudes maliciosas.

Principio de privilegios mínimos: Restringe las cuentas de usuario de la base de datos a solo las operaciones necesarias.

CONCLUSIÓN

Crear una base de datos vulnerable a inyección de SQL con fines educativos puede ser una herramienta valiosa para entender cómo ocurren estas vulnerabilidades y cómo prevenirlas. Al construir un entorno controlado, como una aplicación sencilla que permita inyecciones SQL intencionales, es posible explorar las técnicas que los atacantes usan. En esta aplicación se pueden estudiar las formas para detectar y mitigar estas vulnerabilidades, incluyendo consultas parametrizadas, validación de entradas y cortafuegos de aplicaciones web. Además, comprender estas vulnerabilidades fomenta el desarrollo de aplicaciones más seguras, protegiendo datos sensibles y sistemas críticos frente a posibles ataques.

BIBLIOGRAFÍA

- Belcic, I. (2023, 23 febrero). ¿Qué es la inyección de SQL y cómo funciona? ¿Qué Es la Inyección de SQL y Cómo Funciona? <https://www.avast.com/es-es/c-sql-injection>
- Dizdar, A. (2024, 9 septiembre). Error-Based SQL Injection: Examples and 5 Tips for Prevention. Bright Security. <https://brightsec.com/blog/error-based-sql-injection/>
- Types of SQL Injection? (2020, 15 julio). Acunetix. <https://www.acunetix.com/websitesecurity/sql-injection2/>
- Sgobba, J. P., & Sgobba, J. P. (2024, 10 agosto). Qué es una Inyección SQL (SQL Injection) y cómo solucionarla. Hackmetrix Blog. <https://blog.hackmetrix.com/sql-injection/>
- <https://www.cloudflare.com/es-es/learning/security/threats/how-to-prevent-sql-injection/>