



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

PRACTICA 4.

SEGURIDAD Y VIRTUALIZACION

CARRERA:

INGENIERIA EN SISTEMAS COMPUTACIONALES

GRUPO: 7US

PRESENTA:

RUFINO MENDOZA VAZQUEZ - 21620198

ANA MICHEL LEÓN LEÓN - 21620112

ROSA SALAZAR DOROTEO - 18620216

FERNANDA RUIZ HERAS - 21520151

DOCENTE

ING. EDWARD OSORIO SALINA

Tlaxiaco, Oaxaca, 01 de octubre del 2024.



"Educación, ciencia y tecnología"

Contenido

INTRODUCCIÓN.....	3
DESARROLLO	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
3. REALIZAR PRUEBAS DE INYECCIÓN DE CÓDIGO EN LA APLICACIÓN WEB. ...	8
4. LAS POSIBLES ACCIONES QUE SE PUEDEN REALIZAR PARA PREVENIR ESTE TIPO DE VULNERABILIDADES.	10
CONCLUSIÓN	11
Ilustración 1: Abrir Xampp	4
Ilustración 2: Crear base de datos.....	4
Ilustración 3: Código de base de datos	5
Ilustración 4: Consola de MySQL.....	5
Ilustración 5: Creado ya la base de datos.....	5
Ilustración 6: Código pip install flask mysql-connector-python.....	6
Ilustración 7: Creación de carpeta flask	6
Ilustración 8: Código de flask en python-visual	7
Ilustración 9: Interfaz y base de datos con registro	7
Ilustración 10: Primer producto a buscar y resultado.....	7
Ilustración 11: Segundo producto y resultado	8
Ilustración 12: Tercer producto a buscar y producto.....	8
Ilustración 13: Producto no registrado y no encontrado	8
Ilustración 14: Primera prueba 1OR 1=1.....	9
Ilustración 15: Segunda prueba 2; DROP TABLE productos;	9
Ilustración 16: Tercera prueba 'OR' '1'='1'	9
Ilustración 17: Captura de la inyección	10

INTRODUCCIÓN

La inyección SQL (SQL Injection) es una de las vulnerabilidades más comunes y peligrosas en aplicaciones web que interactúan con bases de datos. Este ataque ocurre cuando un atacante inserta o "inyecta" código malicioso en una consulta SQL a través de un campo de entrada, lo que permite la manipulación o el acceso no autorizado a la base de datos. La gravedad de esta vulnerabilidad radica en que puede dar acceso a información confidencial, modificar datos, eliminar tablas o incluso tomar control completo del servidor de la base de datos. En esta práctica, se busca exponer y explorar esta vulnerabilidad al crear deliberadamente una base de datos susceptible a ataques de inyección SQL. El proceso comienza con la creación de una base de datos simple que contiene una tabla con algunos registros, seguida de una aplicación web básica que permite a los usuarios buscar registros por id, nombre o descripción. La aplicación, al no implementar medidas de seguridad, es vulnerable a inyecciones SQL, lo que permite a los usuarios maliciosos realizar consultas no autorizadas e incluso dañar la base de datos. A través de la experimentación con entradas maliciosas, se podrá observar cómo es posible explotar esta debilidad para extraer datos o ejecutar comandos destructivos. El propósito final es no solo demostrar el peligro de esta vulnerabilidad, sino también aprender cómo prevenirla utilizando técnicas de programación seguras, como consultas preparadas y validación estricta de entradas. Esta práctica es una lección clave en el desarrollo seguro de aplicaciones web, mostrando la importancia de la seguridad desde el diseño inicial del sistema.

PRACTICA 4 – INYECCION SQL

DESARROLLO

1. CREAR UNA BASE DE DATOS CON UNA TABLA QUE CONTENGA AL MENOS 3 REGISTROS.

- Para comenzar a realizar la practica 4 de inyección de SQL, abrimos el panel de control de xampp, iniciamos los modulos de apache y MySQL haciendo clic en los botones “Start” correspondientes.

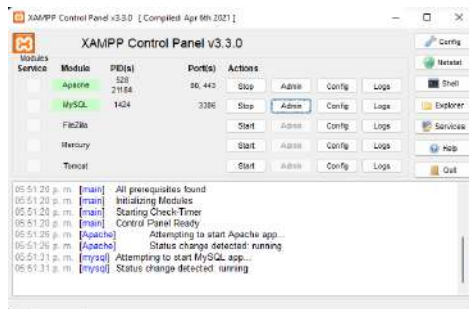


Ilustración 1: Abrir Xampp

- A continuación, abre tu navegador web y navega a <http://localhost/phpmyadmin/>, damos clic en “Nueva” en el panel izquierdo para crear una nueva base de datos con el nombre que nosotros elegiremos en este caso será “mi_base_datos” y posterior a eso le damos crear.

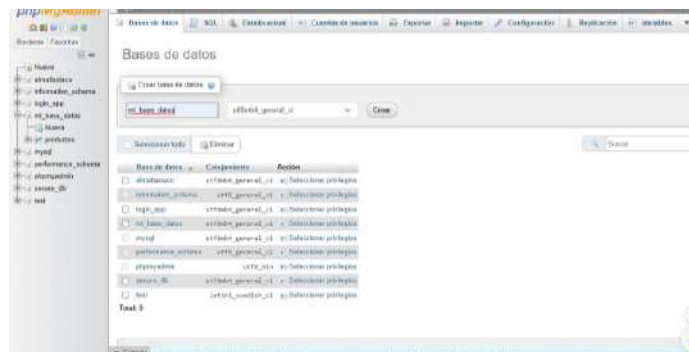


Ilustración 2: Crear base de datos

- Nuestra base de datos obtendrá los siguientes datos que será los campos id, nombre y descripción, el tipo, la longitud, la llave primaria y el autoincremento en el id.

```
CREATE DATABASE mi_base_datos;

-- Usar la base de datos
USE mi_base_datos;

-- Crear la tabla productos
CREATE TABLE productos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  descripcion TEXT
);

-- Insertar datos de ejemplo
INSERT INTO productos (nombre, descripcion) VALUES
('Laptop', 'Computadora portátil'),
('Teléfono', 'Dispositivo móvil'),
('Ratón', 'Dispositivo de entrada'),
('Teclado', 'Dispositivo de entrada');
```

Ilustración 3: Código de base de datos

- Y el código anterior lo agregaremos al sql de phpadmin para crearlo y se agregan algunos registros como se observa en el siguiente ejemplo: id 1, Nombre Laptop, Descripción computadora portátil.



Ilustración 4: Consola de MySQL

- Como observamos ya hemos creado la base correctamente

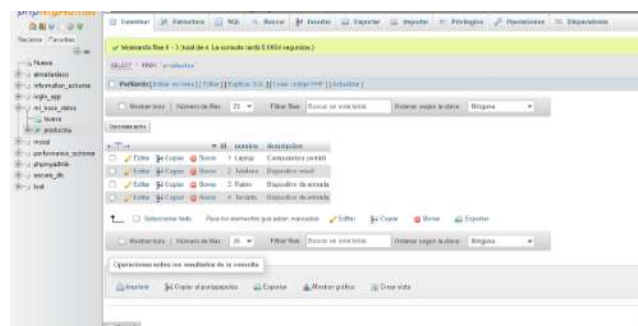
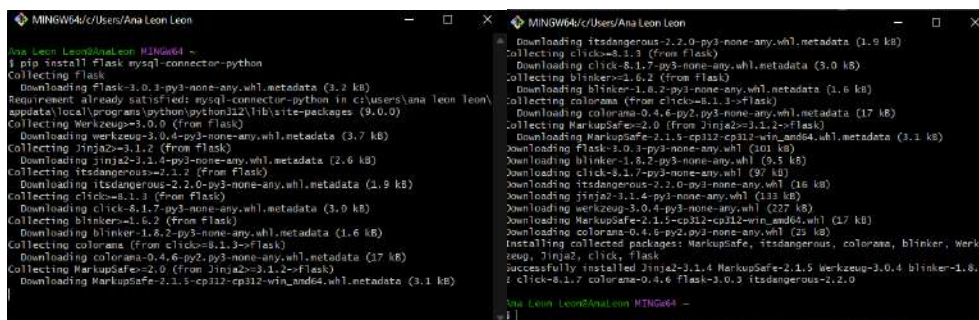


Ilustración 5: Creado ya la base de datos

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.

- Para crear una aplicación web que nos permita buscar un registro por su id, su nombre y su descripción comenzamos instalando el siguiente comando `pip install flask mysql-connector-python` para así poder instalar los paquetes en Python que nos ayudara a conectar una aplicación web con flask a una base de datos MySQL.



```
MINGW64/C:/Users/Ana Leon Leon
Ana Leon Leon@MINGW64: ~
$ pip install flask mysql-connector-python
Collecting flask
  Downloading flask-1.0.3-py3-none-any.whl.metadata (3.2 kB)
Requirement already satisfied: mysql-connector-python in c:\users\ana leon leon\appdata\local\programs\python\python312\lib\site-packages (9.0.0)
Collecting Werkzeug>=3.0.0 (from flask)
  Downloading werkzeug-3.0.4-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Downloading jinja2-3.1.2-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous>=2.1.2 (from flask)
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting click>=8.1.3 (from flask)
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from flask)
  Downloading blinker-1.8.2-py3-none-any.whl.metadata (1.6 kB)
Collecting colorama (from click>=8.1.3->flask)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
  Downloading MarkupSafe-2.1.5-cp12-cp12-win_and64.whl.metadata (3.1 kB)
  Downloading MarkupSafe-2.1.5-cp12-cp12-win_and64.whl (17 kB)
  Downloading flask-3.0.3-py3-none-any.whl (101 kB)
  Downloading blinker-1.8.2-py3-none-any.whl (9.5 kB)
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
  Downloading Jinja2-3.1.4-py3-none-any.whl (133 kB)
  Downloading werkzeug-3.0.4-py3-none-any.whl (227 kB)
  Downloading MarkupSafe-2.1.5-cp12-cp12-win_and64.whl (17 kB)
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: MarkupSafe, itsdangerous, colorama, blinker, Werkzeug, Jinja2, click, flask
Successfully installed Jinja2-3.1.4 MarkupSafe-2.1.5 Werkzeug-3.0.4 blinker-1.8.1 click-8.1.7 colorama-0.4.6 flask-3.0.3 itsdangerous-2.2.0
```

Ilustración 6: Código pip install flask mysql-connector-python

- Después, creamos el archivo de la aplicación Flask el cual se desarrolla una aplicación web, utilizando el framework flask en Python.

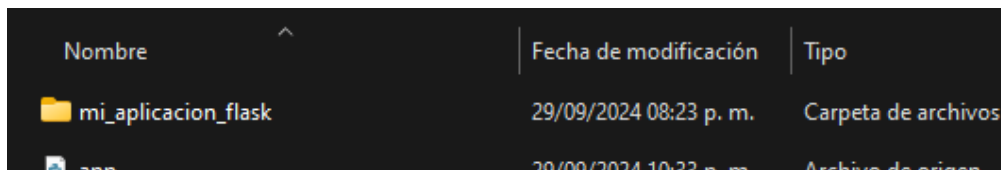


Ilustración 7: Creación de carpeta flask

- Este archivo configura una aplicación web básica que permite a los usuarios buscar productos en una base de datos MySQL mediante un formulario web. Utiliza Flask para manejar las rutas y las interacciones

web, y pymysql para interactuar con la base de datos, utilizamos código css y html para darle forma a nuestra pequeña página.

```

1 from flask import Flask, request, jsonify, render_template
2 import pymysql
3
4 app = Flask(__name__)
5
6 # Configuración de la conexión a la base de datos
7 db = pymysql.connect(
8     host='localhost',
9     user='root',
10    password='',
11    database='db'
12)
13
14 # Crear cursor
15 cursor = db.cursor()
16
17 # Ruta para buscar productos
18 @app.route('/search', methods=['GET'])
19 def search():
20     query = request.args.get('q')
21     if query:
22         cursor.execute(f"SELECT * FROM productos WHERE nombre LIKE '{query}%'")
23         results = cursor.fetchall()
24         return jsonify([{"id": r[0], "nombre": r[1], "descripcion": r[2]} for r in results])
25     return jsonify([])
26
27 if __name__ == '__main__':
28     app.run(debug=True)

```

Ilustración 8: Código de flask en python-visual

- Nos muestra la interfaz que diseñamos donde buscaremos el producto registrado en nuestra base de datos.



Ilustración 9: Interfaz y base de datos con registro

- Comenzamos buscando nuestro primer producto y le agregamos el número 1 y nos arrojará el resultado de la búsqueda que será ID 1, Nombre Laptop, Descripción Computadora portátil.

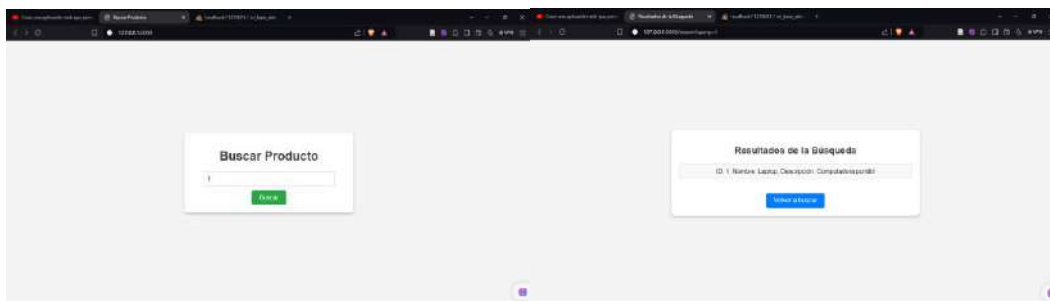


Ilustración 10: Primer producto a buscar y resultado

- Para el siguiente resultado será lo mismo agregamos el número 2 y nos mostrará el segundo resultado: ID 2, Nombre Teléfono, Descripción Dispositivo Móvil.

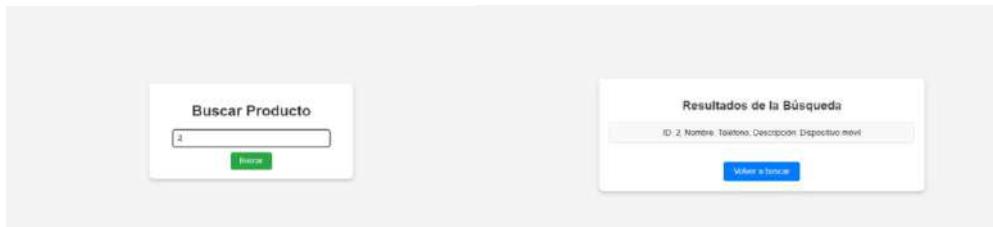


Ilustración 11: Segundo producto y resultado

- Finalmente realizamos la prueba con la tercera prueba y ahora será el número 3 para buscar el siguiente producto lo cual es: ID 3, Nombre Ratón, Descripción Dispositivo de entrada.

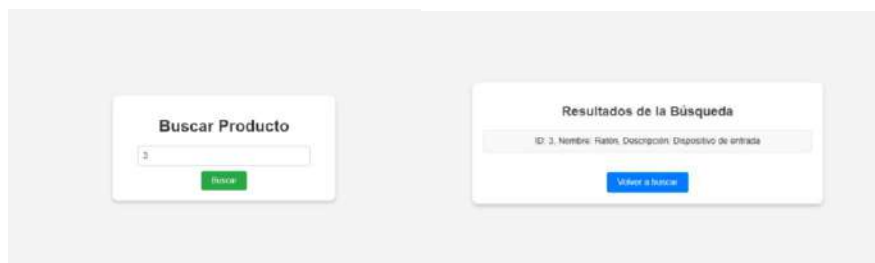


Ilustración 12: Tercer producto a buscar y producto

- Si buscamos un producto el cual no está registrado en nuestra base de datos nos mandara el mensaje de no se encontraron resultados.

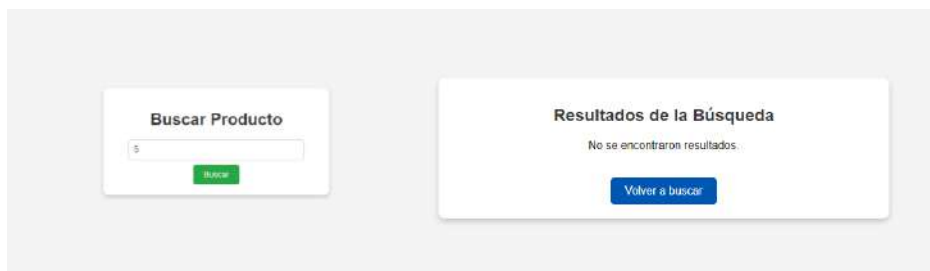


Ilustración 13: Producto no registrado y no encontrado

3. REALIZAR PRUEBAS DE INYECCIÓN DE CÓDIGO EN LA APLICACIÓN WEB.

- Para realizar las pruebas de inyección ingresaremos una consulta maliciosa como el siguiente ejemplo: `1 OR 1=1`, con esta consulta nos devolverá el primer registro de la tabla ya que `1=1` es verdadero y la parte restante de la consulta se ignora.

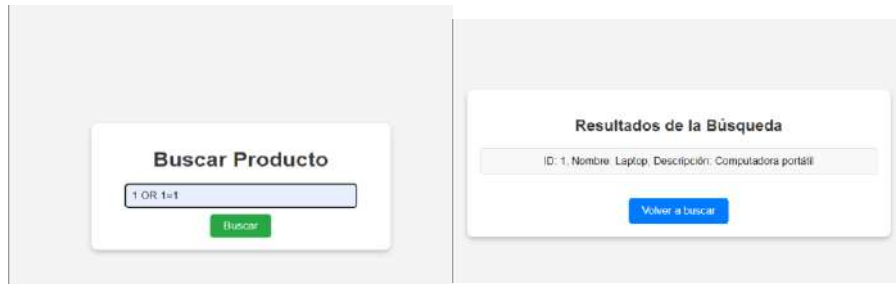


Ilustración 14: Primera prueba 1OR 1=1

- La segunda prueba de Inyección SQL para eliminar la tabla será **2; DROP TABLE productos;** En esta consulta intenta eliminar la tabla de productos, si el usuario tiene los permisos para modificar la estructura la tabla será eliminada.

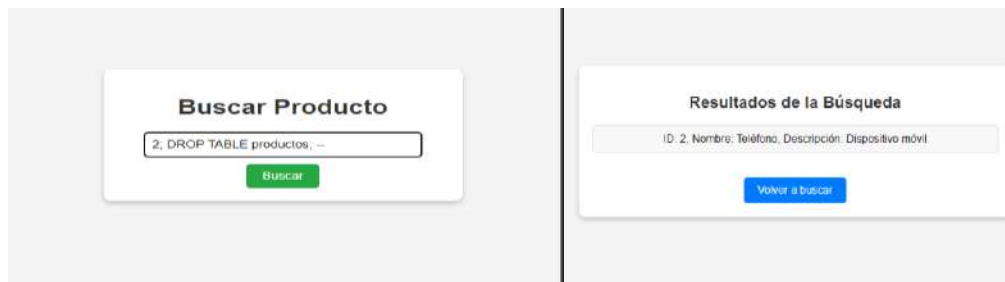


Ilustración 15: Segunda prueba 2; DROP TABLE productos;

- Para la última prueba ejecutaremos 'OR' '1'='1' este comando hará que muestre la vulnerabilidad de los registros y nos mostrara todos los registros en nuestra tabla.

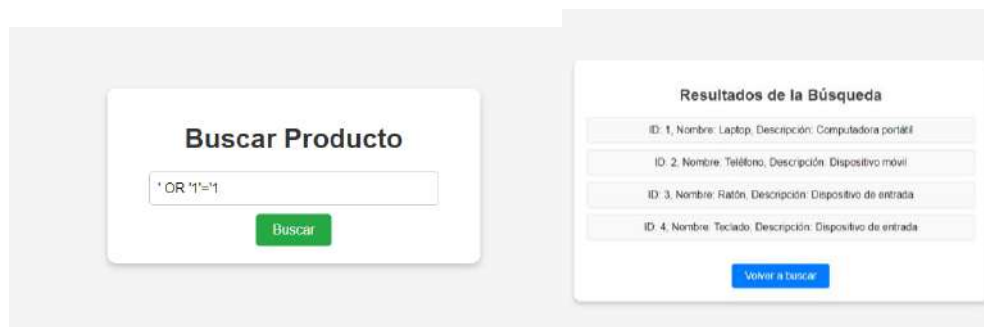


Ilustración 16: Tercera prueba 'OR' '1'='1'

- La siguiente ilustración nos indica que nuestra aplicación Flask está recibiendo intentos de inyección SQL.

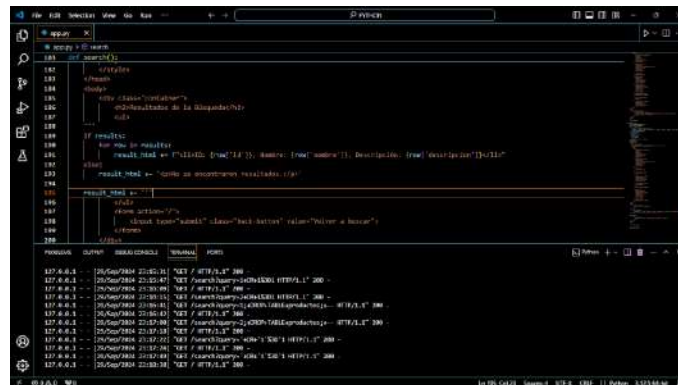


Ilustración 17: Captura de la inyección

4. LAS POSIBLES ACCIONES QUE SE PUEDEN REALIZAR PARA PREVENIR ESTE TIPO DE VULNERABILIDADES.

- Usar consultas parametrizadas (Prepared Statements).
- Implementar procedimientos almacenados en la base de datos.
- Implementar protección contra Cross-Site Scripting (XSS).
- Almacenar las configuraciones sensibles usando variables de entorno.
- Aplicar límites de tasa (Rate Limiting) para prevenir abusos.

CONCLUSIÓN

La inyección de SQL es una vulnerabilidad crítica que, si no se aborda correctamente, puede comprometer seriamente la seguridad de una aplicación web. A través de esta práctica, se ha demostrado cómo entradas no controladas pueden permitir la manipulación de datos y la ejecución de comandos maliciosos. Para mitigar estos riesgos, es esencial implementar prácticas de seguridad como la validación de entradas, el uso de consultas preparadas y procedimientos de codificación seguros. Además, es importante conocer las distintas variantes de inyección SQL, como las basadas en errores y en tiempo, para anticiparse a posibles ataques.