TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

SEGURIDAD Y VIRTUALIZACIÓN

Practica 4 - Inyeccion SQL

Integrantes del Equipo:

Arnol Jesus Cruz Ortiz

Amilkar Vladimir Reyes Reyes

Rael Gabriel Bautista

Sandra Gabriela Velasco Guzmán

Docente:

Edward Osorio Salinas

Carrera:

Ingeniera en Sistemas Computacionales

Grupo: 7US

Semestre: Agosto – diciembre 2024

26/Septiembre/2024

PRACTICA 4 - INYECCION SQL

Objetivo

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

Desarrollo

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

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.

Realizar pruebas de inyección de código en la aplicación web.

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

Para el desarrollo de esta práctica usaremos Python, flask y SQLlite3.

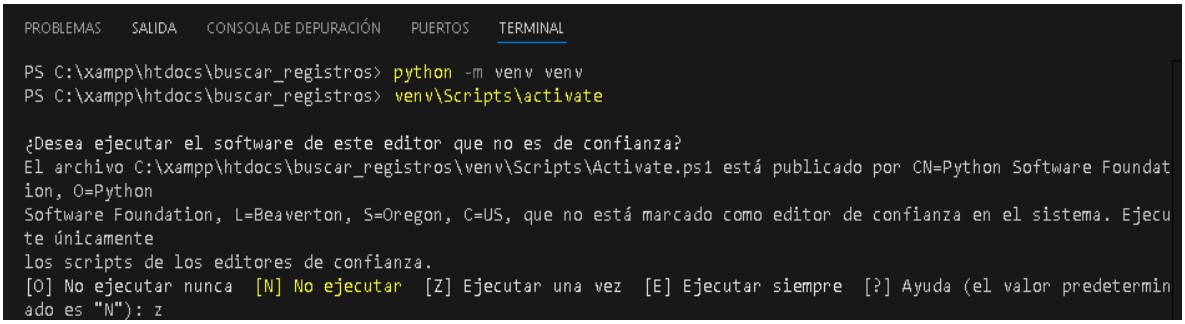
Y como bien sabemos Python proporciona el lenguaje base para el desarrollo de todo tipo de aplicaciones.

Uttilizxaremos Flask es un framework web ligero para Python, útil para crear aplicaciones web de forma rápida y con una complejidad mínima. Flask proporciona las herramientas esenciales para crear rutas web, manejar solicitudes de usuarios, renderizar plantillas y administrar sesiones. Flask que permite crear aplicaciones web rápidas y sencillas con Python.

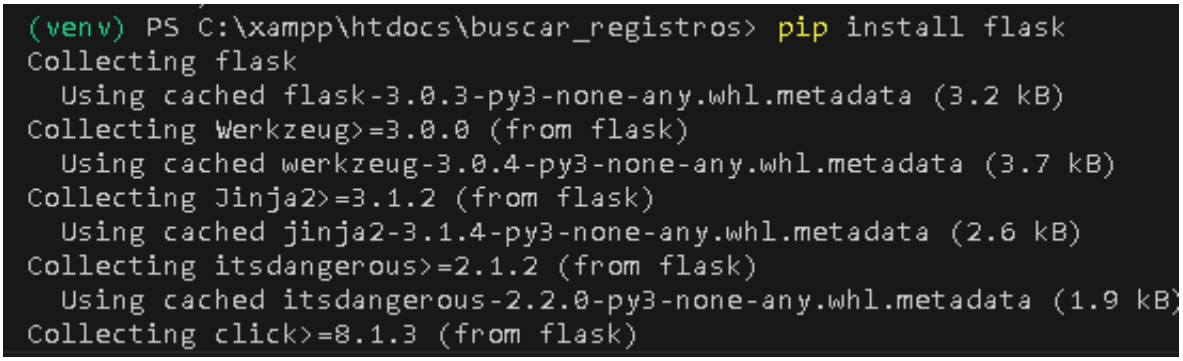
Y también utilizaremos SQLite3 que es un sistema de gestión de bases de datos relacionales que almacena datos en un solo archivo en el disco. Está integrado, lo que significa que no requiere un servidor separado para funcionar, lo que lo hace ideal para aplicaciones locales o proyectos web más pequeños. También ofrece una solución de base de datos ligera y fácil de manejar para almacenar información en tus aplicaciones Python y Flask.

“INICIO DE APLICACIÓN”

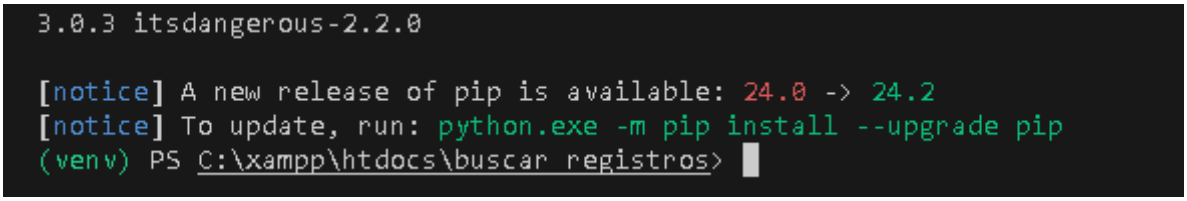
Para empezar crearemos un entorno virtual Python con: python -m venv venv , en nuestra terminal en la siguiente dirección: PS C:\xampp\htdocs\buscar_registros> y activaremos el evento con: venv\Scripts\activate, nos preguntara si queremos ejecutar el software, a lo que daremos ejecutar una vez [z].



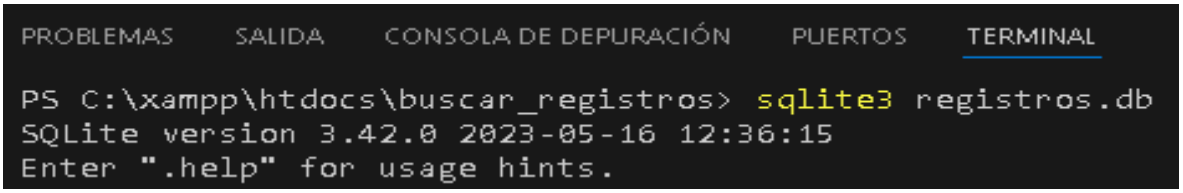
Una vez activado el entorno Instalaremos los módulos de flask para poder usarlos en Python. Con el siguiente comando: pip install flask



Esperaremos a que se instale flask.



Utilizaremos SQLlite3 Registros.db, dicho comando lo ejecutaremos en la carpeta PS C:\xampp\htdocs\buscar_registros> Esto nos creara una base de datos con el nombre de registros.



Ahora crearemos una tabla llamada registros, estando en la terminal sqlite> CREATE TABLE registros (id INTEGER PRIMARY KEY, nombre TEXT, descripción TEXT); Insertamos 5 valores como ejemplo de registros almacenador en la tabla.

INSERT INTO registros (nombre, descripcion) VALUES ('Ejemplo 1', 'Descripción 1'); INSERT INTO registros (nombre, descripcion) VALUES ('Ejemplo 5', 'Descripción 5');

Para salir del entorno escribimos. exit

```
sqlite> CREATE TABLE registros (id INTEGER PRIMARY KEY, nombre TEXT, descripcion TEXT);
sqlite> INSERT INTO registros (nombre, descripcion) VALUES ('Ejemplo 1', 'Descripción 1');
sqlite> INSERT INTO registros (nombre, descripcion) VALUES ('Ejemplo 2', 'Descripción 2');
sqlite> INSERT INTO registros (nombre, descripcion) VALUES ('Ejemplo 3', 'Descripción 3');
sqlite> INSERT INTO registros (nombre, descripcion) VALUES ('Ejemplo 4', 'Descripción 4');
sqlite> INSERT INTO registros (nombre, descripcion) VALUES ('Ejemplo 5', 'Descripción 5');
```

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. Este es un ejemplo.

```python
# 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)
    #  - Buscar por name = ' OR 1=1 --Esto devolverá todos los registros de la tabla (lo que no queremos
    # - Buscar por name = ' OR 1=1; DROP TABLE table; -- Esto eliminará la tabla (lo que no queremos), e
    result = c.fetchall()
    conn.close()
    return str(result)

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

Este código usando Python, nos permitirá hacer una conexión a la base de datos registros.db y hacer consultas no seguras. Con html para visualizar los registros que estamos haciendo.

inyeccion.py > ...

```python
import sqlite3
from flask import Flask, request, render_template_string

app = Flask(__name__)


@app.route("/")
def index():
    html = """
    <form action="/search" method="get">
        <input type="text" name="query" placeholder="Buscar">
        <input type="submit" value="Buscar">
    </form>
    <div>
        <table border="1">
            <tr>
                <th>ID</th>
                <th>Nombre</th>
                <th>Descripción</th>
            </tr>
            {% for row in registros %}
                <tr>
                    <td>{{ row[0] }}</td>
                    <td>{{ row[1] }}</td>
                    <td>{{ row[2] }}</td>
                </tr>
            {% endfor %}
        </table>
    </div>
    """
```

```python
    # Conectarse a la base de datos y obtener todos los registros
    conn = sqlite3.connect("registros.db")
    c = conn.cursor()
    c.execute("SELECT id, nombre, descripcion FROM registros ORDER BY id DESC")
    registros = c.fetchall()  # Obtiene todos los registros
    conn.close()

    # Renderiza la plantilla HTML
    return render_template_string(html, registros=registros)


@app.route("/search")
def search():
    query = request.args.get("query")
    conn = sqlite3.connect("registros.db")
    c = conn.cursor()

    # Consulta segura usando parámetros para evitar inyecciones SQL
    # c.execute(
    #    "SELECT * FROM registros WHERE id = ? OR nombre = ? OR descripcion = ?",
    #   (query, query, query),
    # )

    # Consulta no segura usando parámetros para evitar inyecciones SQL
    c.execute(
        f"SELECT * FROM registros WHERE id = '{query}' OR nombre = '{query}' OR descripcion = '{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
    #  - Buscar por name = ' OR 1=1 --Esto devolverá todos los registros de la tabla (lo que no queremos), ya
    # - Buscar por name = ' OR 1=1; DROP TABLE table; -- Esto eliminará la tabla (lo que no queremos), esto s

    registros = c.fetchall()
    conn.close()
```

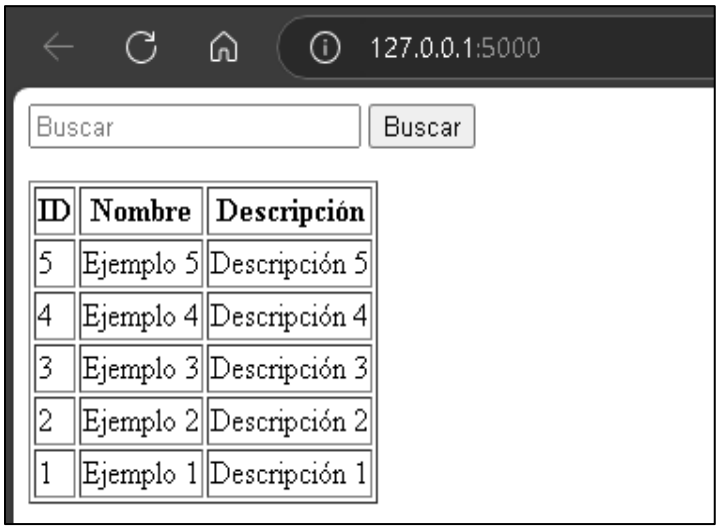
En cada consulta de una inyección nos mostrara los cambios que realiza en el momento.

```python
    # resultados de la búsqueda
    html = """
    <form action="/search" method="get">
        <input type="text" name="query" placeholder="Buscar" value="{{ query }}">
        <input type="submit" value="Buscar">
    </form>
    <div>
        <table border="1">
            <tr>
                <th>ID</th>
                <th>Nombre</th>
                <th>Descripción</th>
            </tr>
            {% for row in registros %}
                <tr>
                    <td>{{ row[0] }}</td>
                    <td>{{ row[1] }}</td>
                    <td>{{ row[2] }}</td>
                </tr>
            {% endfor %}
        </table>
    </div>
    """

    # Renderiza los resultados de la búsqueda
    return render_template_string(html, registros=registros, query=query)

if __name__ == "__main__":
    app.run(
        debug=True
    )  # Activar el modo de depuración para ver más detalles de los errores
```

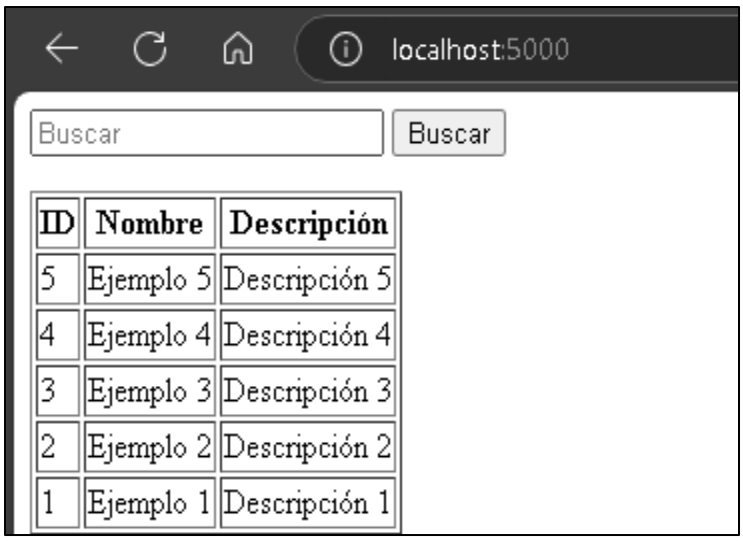
Una vez terminado el código lo ejecutaremos de la siguiente forma

```
PS C:\xampp\htdocs\buscar_registros> python inyeccion.py
 * Serving Flask app 'inyeccion'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server inste
ad.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 145-916-650
```

El código nos dará un directorio http://127.0.0.1:5000 que es el host local



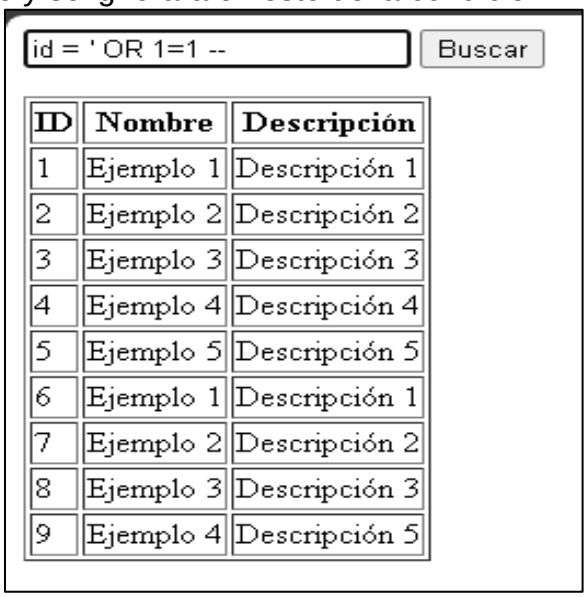
También podemos abrirlo con localhost:5000, lo que vemos es una vista como ejemplo de lo que contiene la tabla.



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

Ahora realizaremos las búsquedas, primero con el id, escribiendo en el buscador la siguiente inyección id = 1 OR 1=1 --.

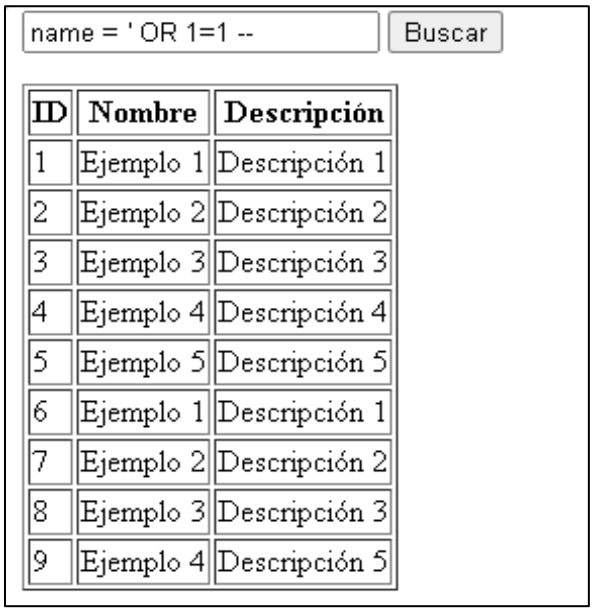
Esto devolverá todos los registros de la tabla (lo que no queremos), ya que 1=1 siempre es verdadero y se ignorará el resto de la condición



ID	Nombre	Descripción
1	Ejemplo 1	Descripción 1
2	Ejemplo 2	Descripción 2
3	Ejemplo 3	Descripción 3
4	Ejemplo 4	Descripción 4
5	Ejemplo 5	Descripción 5
6	Ejemplo 1	Descripción 1
7	Ejemplo 2	Descripción 2
8	Ejemplo 3	Descripción 3
9	Ejemplo 4	Descripción 5

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



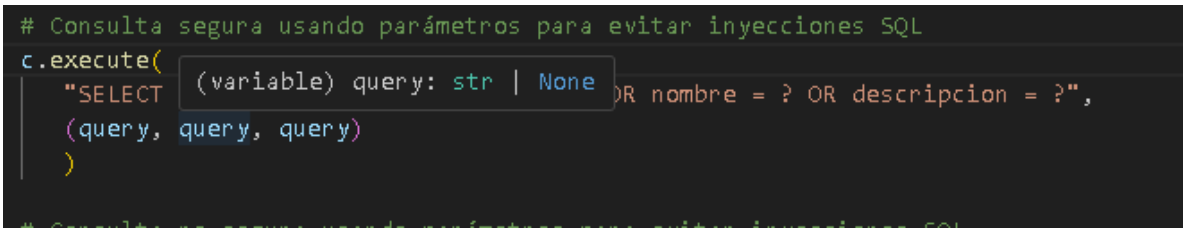
ID	Nombre	Descripción
1	Ejemplo 1	Descripción 1
2	Ejemplo 2	Descripción 2
3	Ejemplo 3	Descripción 3
4	Ejemplo 4	Descripción 4
5	Ejemplo 5	Descripción 5
6	Ejemplo 1	Descripción 1
7	Ejemplo 2	Descripción 2
8	Ejemplo 3	Descripción 3
9	Ejemplo 4	Descripción 5

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. Siendo este el mas peligroso si no tenemos buenas practicas

Para que no suceda eso, debemos hacer buenas consultas y con eso terminamos.



CONCLUSIÓN

En esta práctica, se implementó una vulnerabilidad de inyección de código SQL en una aplicación web. La falta de validación adecuada de entradas por parte del usuario permitió modificar la consulta SQL, haciendo que se ejecuten instrucciones no previstas por el sistema. Esta vulnerabilidad puede ser explotada para acceder a información confidencial o realizar acciones no autorizadas.

Lecciones Aprendidas:

1. La inyección SQL ocurre cuando los datos ingresados por el usuario no están debidamente sanitizados y se utilizan directamente en consultas SQL.
2. Para prevenir este tipo de ataques, se deben utilizar consultas preparadas o declaraciones parametrizadas en lugar de concatenar directamente los valores ingresados en la consulta.
3. Es crucial implementar validación de datos y controles de seguridad en cada entrada de usuario.

Este ejercicio resalta la importancia de diseñar sistemas robustos que no dependan únicamente de la confianza en los datos que provienen de los usuarios.