Programación en Bases de Datos

Proyecto Hacking Ético: SQL Injection

Alejandro Bofill Gonzalez - David Gordillo Burrero - Aarón Manzano Mateos - Ángel Macías Barro

Índice de contenidos

1. Introducción al proyecto

El presente documento aborda el tema del hacking ético y su aplicación en el estudio de vulnerabilidades en bases de datos, con un enfoque particular en las inyecciones SQL. Estas vulnerabilidades constituyen uno de los riesgos más comunes y críticos en aplicaciones web que interactúan con bases de datos, permitiendo a los atacantes manipular o acceder a información de manera no autorizada. Este trabajo tiene como propósito profundizar en la comprensión de las inyecciones SQL, analizando sus tipos, los riesgos asociados y los motivos por los cuales ocurren, para fomentar una perspectiva integral de la seguridad en el desarrollo de software. Para facilitar el aprendizaje práctico y la evaluación de estas vulnerabilidades, se ha desarrollado un laboratorio interactivo basado en tecnologías como Flask y Python. Este entorno de simulación permite experimentar con diferentes tipos de inyecciones SQL, ofreciendo a los usuarios una experiencia inmersiva y controlada. El laboratorio incluye aplicaciones web diseñadas específicamente para emular escenarios reales de inyecciones SQL, permitiendo realizar pruebas tanto en bases de datos Oracle como en PostgreSQL. Estas bases de datos, seleccionadas por su relevancia y amplio uso en entornos empresariales, proporcionan un contexto diverso y representativo para explorar las técnicas de ataque y sus consecuencias.

Cada tipo de inyección SQL implementada en el laboratorio ha sido cuidadosamente seleccionada para cubrir un amplio espectro de vulnerabilidades. Entre ellas, se encuentran las inyecciones basadas en errores, que explotan los mensajes de error generados por la base de datos para extraer información sensible; las inyecciones de tipo Ünion Attack", que utilizan la cláusula UNION para combinar resultados de consultas y obtener datos confidenciales; las inyecciones basadas en booleanos, que permiten inferir información mediante la manipulación de condiciones lógicas; y las inyecciones ciegas, que aprovechan diferencias en el comportamiento de la aplicación para deducir detalles de la base de datos sin generar mensajes de error explícitos.

El diseño del laboratorio incluye páginas individuales para cada tipo de inyección, con formularios de inicio de sesión vulnerables, credenciales específicas para realizar las pruebas y explicaciones detalladas sobre el ataque correspondiente. Esto permite que los usuarios comprendan tanto la teoría subyacente como la ejecución práctica de cada tipo de inyección. Además, el laboratorio cuenta con documentación complementaria que explica cómo se implementaron las vulnerabilidades y su relevancia en entornos reales, proporcionando una base sólida para que los participantes puedan identificar y mitigar estos riesgos en sus propios proyectos.

Este trabajo no solo subraya los riesgos asociados con las inyecciones SQL, como el acceso no autorizado a datos confidenciales, la alteración o eliminación de información crítica y el compromiso total del servidor de base de datos, sino que también explora las causas más comunes detrás de estas vulnerabilidades. Entre ellas, se encuentran la falta de sanitización de las entradas del usuario, el uso de consultas dinámicas inseguras y la ausencia de auditorías de seguridad durante el desarrollo de software. Al comprender estos factores, se busca fomentar la adopción de mejores prácticas en la construcción de aplicaciones seguras y resilientes.

En conclusión, este proyecto combina un enfoque teórico y práctico para abordar una de las amenazas más relevantes en el ámbito de la seguridad informática. Al proporcionar un entorno interactivo y educativo, se espera que este laboratorio no solo aumente la conciencia sobre la importancia de prevenir las inyecciones SQL, sino que también capacite a los participantes en la identificación, explotación controlada y mitigación de estas vulnerabilidades. Este esfuerzo

refleja el compromiso con la promoción de un desarrollo de software más seguro, alineado con los principios del hacking ético y la ciberseguridad.

2. Inyección SQL

2.1. ¿Qué es una inyección SQL?

Imagina que estás en un restaurante y entregas tu orden al mesero. Si todo funciona como debería, el mesero simplemente transmite tu pedido al chef, quien prepara la comida según lo solicitado. Ahora bien, ¿qué pasaría si, en lugar de un pedido normal, decides escribir en la nota algo como: "Quiero una pizza, y además dame acceso a la caja registradora¿ Si el sistema del restaurante no tiene medidas para validar las órdenes, es posible que el mensaje llegue al chef, quien podría malinterpretarlo como una instrucción válida y permitirte hacer algo que no deberías poder hacer. Algo similar ocurre con una inyección SQL, pero en el contexto de aplicaciones web y bases de datos.

Una inyección SQL es un tipo de ataque en el que un atacante manipula las consultas que una aplicación web hace a su base de datos para ejecutar comandos maliciosos. Esto sucede cuando el sistema no valida adecuadamente las entradas proporcionadas por los usuarios y las trata directamente como parte de la consulta SQL. Por ejemplo, en una página de inicio de sesión, si un usuario malintencionado introduce un texto diseñado específicamente para alterar la lógica de la consulta SQL que valida las credenciales, podría obtener acceso no autorizado al sistema sin necesidad de conocer la contraseña.

Un caso real que ilustra el impacto de las inyecciones SQL ocurrió en el año 2012, cuando un grupo de atacantes explotó esta vulnerabilidad en una aplicación de LinkedIn. Mediante una inyección SQL, los atacantes lograron acceder a información confidencial de usuarios, incluidos datos de inicio de sesión. Este incidente no solo expuso la información personal de millones de personas, sino que también dañó la reputación de la compañía y resaltó la gravedad de no implementar medidas de seguridad adecuadas.

Por ejemplo, imagina una consulta SQL en una aplicación web que maneja la autenticación de usuarios. Una consulta típica podría ser:

```
SELECT * FROM usuarios
WHERE nombre_usuario = 'usuario'
AND contraseña = 'contraseña';
```

Si el sistema permite que un atacante ingrese algo como `usuario' OR '1'='1` en lugar del nombre de usuario, la consulta resultante sería:

```
SELECT * FROM usuarios
WHERE nombre_usuario = 'usuario' OR '1'='1'
AND contraseña = 'contraseña';
```

La condición `'1'='1'` siempre es verdadera, lo que significa que el atacante podría acceder al sistema sin necesidad de proporcionar credenciales válidas. Este ejemplo muestra cómo una entrada maliciosa puede alterar la lógica de una consulta SQL, dándole al atacante el control.

Las inyecciones SQL pueden tener consecuencias graves, que incluyen el robo de datos confidenciales, la modificación o eliminación de registros, y en casos extremos, el control total del

servidor de la base de datos. Además, son una de las vulnerabilidades más comunes en aplicaciones web, según el *OWASP* (Open Web Application Security Project). Sin embargo, prevenirlas no es complicado si se aplican prácticas adecuadas, como el uso de consultas preparadas, la validación estricta de entradas y la implementación de controles de acceso robustos.

En resumen, una inyección SQL no es solo un fallo técnico, sino un ejemplo de cómo pequeñas omisiones en la seguridad pueden tener grandes repercusiones. Conocer cómo ocurren y cómo prevenirlas es esencial para cualquier profesional que desarrolle aplicaciones conectadas a bases de datos, ya que no solo se trata de proteger sistemas, sino también la confianza y la información de los usuarios.

2.2. Causa principal de las inyecciones SQL

3. Preparación del entorno

En este apartado se detallarán los requerimientos necesarios para la correcta ejecución del proyecto, así como las funciones que se han implementado para la creación de la base de datos y la inserción de datos en la misma.

3.1. Prerequisitos

A modo de base para el correcto desarrollo y ejecución del proyecto, es necesario tener instalado en el sistema los siguientes paquetes:

- Oracle Database
- PostgreSQL Database
- Python

La version descargada de los anteriores paquetes puede ser la que ha sido configurada en anteriores prácticas desarrolladas en la asignatura de Programación en Bases de Datos.

3.2. Requerimientos de ejecución

Una vez configurados correctamente los prerequisitos que no estan ligados especificamente con esta práctica, se procederá a la instalación de las librerias y frameworks de python que han sido necesarios para el desarrollo de este laboratorio. Para facilitar este proceso se ha generado un fichero con las librerias necesarias que se puede instalar mediante el siguiente comando:

```
1    pip install -r requirements.txt
```

En dicho fichero *requirements.txt* se encuentran las siguientes librerias:

- **Flask** con version 2.2.0
- **werkzeug** con version 2.2.0
- **oracledb** con version mayor o igual a 2.4.1
- **psycopg2** con version mayor o igual a 2.9.9
- **requests** con version mayor o igual a 2.32.2

- **termcolor** con version 2.2.0
- **yaspin** con version mayor o igual a 3.1.0

4. Conexión de la base de datos

Para la conexión de la base de datos se han implementado funciones de conectar y desconectar que permiten la conexión a Oracle y PostgreSQL. Estas funciones se han implementado en los archivos *setupOracle.py* y *setupPostgres.py*. A continuación se detallan las funciones implementadas en cada uno de los archivos.

4.1. Oracle

A continuación se muestran las funciones que se ha implementado en el archivo *setupOracle.py* para la conexión a Oracle.

```
# Función para conectar a la base de datos
def dbConectarOracle():
    ip = "localhost"
    puerto = 1521
    s_id = "xe"
    usuario = "system"
    contrasena = "12345"

    print("---dbConectarOracle---")
    print("---Conectando a Oracle---")

    try:
        conexion = PBD.connect(user=usuario, password=contrasena, host=
           ip, port=puerto, sid=s_id)
        print("Conexión realizada a la base de datos", conexion)
        return conexion
    except PBD.DatabaseError as error:
        print("Error en la conexión")
        print(error)
        return None

# Función para desconectar de la base de datos
def dbDesconectar(conexion):
    print("---dbDesconectar---")
    try:
        if conexion:  # Verifica que la conexión no sea None
            conexion.commit()  # Confirma los cambios

            conexion.close()
            print("Desconexión realizada correctamente")
            return True
        else:
            print("No hay conexión para cerrar.")
            return False
    except PBD.DatabaseError as error:
        print("Error en la desconexión")
        print(error)
```

```
37         return False
```

4.2. PostgreSQL

A continuación se muestra la funciones que se ha implementado en el archivo *setupPostgres.py* para la conexión a PostgreSQL.

```
1
2  def dbConectarPostgreSQL():
3  ip = "localhost"
4  puerto = 5432
5  basedatos = "Empresa"
6
7  usuario = "postgres"
8  contrasena = "12345"
9
10 print("---dbConectarPostgreSQL---")
11 print("---Conectando a Postgresql---")
12
13 try:
14     conexion = PBD.connect(user=usuario, password=contrasena, host=ip,
          port=puerto, database=basedatos)
15     print("Conexión realizada a la base de datos",conexion)
16     return conexion
17 except PBD.DatabaseError as error:
18     print("Error en la conexión")
19     print(error)
20     return None
21
22 #-------------------------------------------------------------------
23
24 def dbDesconectar(conexion):
25 print("---dbDesconectar---")
26 try:
27     conexion.commit()  # Confirma los cambios
28     conexion.close()
29     print("Desconexión realizada correctamente")
30     return True
31 except PBD.DatabaseError as error:
32     print("Error en la desconexión")
33     print(error)
34     return False
```

5. Creación de tablas

Para la creación de las tablas en Oracle y PostgreSQL se han implementado dos funciones que permiten la creación de las mismas. Se ha decidido crear una tabla llamada *Usuarios* con los campos *id*, *username*, *password* y *session_ cookie*. Además, se ha incluido la inserción de usuarios de ejemplo en la tabla, para poder hacer uso de ellos en las inyecciones del laboratorio. Estas funciones se han implementado en los archivos *setupOracle.py* y *setupPostgres.py*. A continuación se detallan las funciones implementadas en cada uno de los archivos.

5.1. Oracle

A continuación se muestra la función que se ha implementado en el archivo *setupOracle.py* para la creación de la tabla en Oracle.

```
    # Funcion para la configuracion de tablas
    def configuracionTablas\_oracle(conexion):
        print("---configuracionTablas---")
        try:
            cursor = conexion.cursor()

            # Crear tabla Usuarios si no existe con columna session\
               _cookie
            consulta = """
                BEGIN
                    EXECUTE IMMEDIATE 'CREATE TABLE Usuarios (
                        id NUMBER GENERATED BY DEFAULT AS IDENTITY
                           PRIMARY KEY,
                        username VARCHAR2(50) NOT NULL UNIQUE,
                        password VARCHAR2(50) NOT NULL,
                        session\_cookie VARCHAR2(255)
                    )';
                EXCEPTION
                    WHEN OTHERS THEN
                        IF SQLCODE = -955 THEN
                            NULL;  -- Ignora si la tabla ya existe
                        ELSE
                            RAISE;
                        END IF;
                END;
            """
            cursor.execute(consulta)

            # Insertar usuarios de ejemplo solo si la tabla esta vacia
            cursor.execute("SELECT COUNT(*) FROM Usuarios")
            count = cursor.fetchone()[0]
            print("Usuarios en la tabla:", count)
            if count == 0:
                usuarios\_ejemplo = [
                    ("admin", "password123", "
                       t4SpnpWyg76A3K2BqcFh2vODq0fqJGvs38ydh9"),
                    ("user1", "password1", "
                       d382yd8n21df4314fn817yf6834188ls023d8d"),
                    ("user2", "password2", "
                       u73dv226d726gh23fnjncuyg0q9udfjf47eueu")
                ]
                cursor.executemany(
                    "INSERT INTO Usuarios (username, password, session\
                       _cookie) VALUES (:username, :password, :session\
                       _cookie)",
                    usuarios\_ejemplo
                )
                print("Usuarios de ejemplo insertados correctamente.")
            else:
```

```
                print("La tabla Usuarios ya contiene datos.")

            cursor.close()
            print("Tabla 'Usuarios' creada o verificada exitosamente")
            return True
        except PBD.DatabaseError as error:
            print("Error al crear la tabla o insertar usuarios")
            print(error)
            return False
```

5.2. PostgreSQL

A continuación se muestra la función que se ha implementado en el archivo *setupPostgres.py* para la creación de la tabla en PostgreSQL.

```
def configuracion_tablas_postgresql(conexion):
print("---configuracion_tablas_postgresql---")
try:
    cursor = conexion.cursor()

    # Crear la tabla Usuarios si no existe con columna session_cookie
    consulta = """
        CREATE TABLE IF NOT EXISTS Usuarios (
            id SERIAL PRIMARY KEY,
            username VARCHAR(50) NOT NULL UNIQUE,
            password VARCHAR(50) NOT NULL,
            session_cookie VARCHAR(255)
        );
    """
    cursor.execute(consulta)

    # Insertar usuarios de ejemplo solo si la tabla esta vacia
    cursor.execute("SELECT COUNT(*) FROM Usuarios")
    count = cursor.fetchone()[0]
    if count == 0:
        usuarios_ejemplo = [
            ("admin", "password123", "
               t4SpnpWyg76A3K2BqcFh2vODq0fqJGvs38ydh9"),
            ("user1", "password1", "
               d382yd8n21df4314fn817yf6834188ls023d8d"),
            ("user2", "password2", "
               u73dv226d726gh23fnjncuyg0q9udfjf47eueu")
        ]
        cursor.executemany(
            "INSERT INTO Usuarios (username, password, session_cookie)
               VALUES (%s, %s, %s)",
            usuarios_ejemplo
        )
        print("Usuarios de ejemplo insertados correctamente.")
    else:
        print("La tabla Usuarios ya contiene datos.")

    cursor.close()
```

```
35      print("Tabla 'Usuarios' creada o verificada exitosamente en
            PostgreSQL")
36      return True
37  except PBD.DatabaseError as error:
38      print("Error al crear la tabla o insertar usuarios en PostgreSQL")
39      print(error)
40      return False
```

6. Introducción al laboratorio

Se ha desarrollado un laboratorio interactivo que permite experimentar con diferentes tipos de inyecciones SQL en bases de datos Oracle y PostgreSQL. En los siguientes apartados se explicará en detalle cada inyección y cómo se ha implementado en el laboratorio.

7. (

Inyección basada en errores de la base de datos) La **inyección SQL basada en errores** es un tipo de ataque de inyección SQL en el que el atacante explota la información expuesta directamente por los mensajes de error generados por la base de datos. Estos mensajes de error proporcionan pistas sobre la estructura, el esquema y el contenido de la base de datos, lo que permite al atacante ejecutar consultas maliciosas para obtener acceso no autorizado a los datos o comprometer la integridad del sistema.

7.1. Mecanismo del Ataque

- **Explotación de Mensajes de Error:** Durante el desarrollo de aplicaciones web, los desarrolladores suelen habilitar mensajes de error detallados para depurar problemas. Si estos mensajes no se desactivan en producción, los atacantes pueden intencionalmente introducir consultas mal formadas o comandos SQL en los puntos de entrada de la aplicación (por ejemplo, formularios o parámetros URL) para provocar errores.
- **Uso de Consultas Maliciosas:** El atacante inserta código SQL diseñado para causar un error deliberado y obtener un mensaje detallado de la base de datos. Estos mensajes pueden revelar información sensible como nombres de tablas, nombres de columnas, tipos de datos o incluso fragmentos del contenido almacenado.
- **Iteración del Proceso:** Basándose en los datos recopilados de los mensajes de error, el atacante ajusta y refina sus consultas maliciosas para extraer información adicional o lograr un acceso más profundo al sistema.

Para acceder a la sección especifica para esta inyección en el laboratorio, una vez desplegado el servidor con una apariencia como la siguiente:

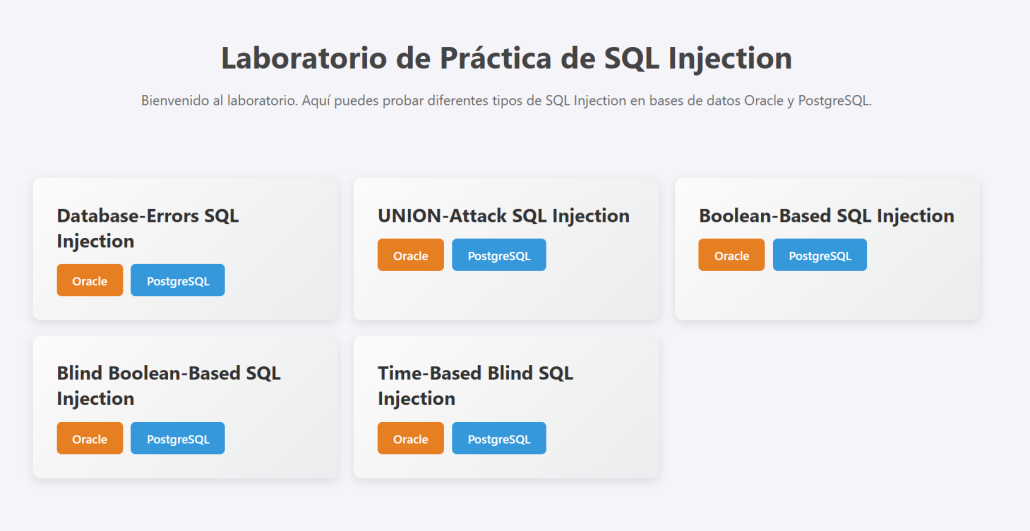

Figura 1: Página de inicio del laboratorio de inyecciones SQL

8. (

Inyección basada en Union Attack) La inyección SQL basada en **UNION** es una técnica en la que un atacante utiliza la cláusula UNION para combinar los resultados de una consulta legítima con datos maliciosamente solicitados, permitiendo extraer información sensible de la base de datos. Para llevar a cabo este ataque, el atacante identifica puntos vulnerables en la aplicación web, determina el **número de columnas** en la consulta original y luego inyecta una consulta maliciosa que utiliza **UNION SELECT** para unir los resultados deseados. Para prevenir este tipo de ataques, es esencial validar y sanear todas las entradas de usuario, utilizar consultas parametrizadas y aplicar el principio de privilegios mínimos en las cuentas de la base de datos.

8.1. Proceso del ataque

1. **Identificación de puntos vulnerables:** El atacante busca parámetros de entrada en la aplicación web que interactúan directamente con la base de datos. Esto puede incluir campos de formularios, parámetros en la URL, cookies o encabezados HTTP.

2. **Determinación del número de columnas:** Utilizando inyecciones como `ORDER BY` o consultas de prueba con `UNION SELECT NULL`, el atacante descubre el número de columnas en la consulta original. Esto asegura que la consulta maliciosa inyectada sea compatible con la estructura de la consulta legítima.

3. **Construcción de la inyección:** Una vez identificados los parámetros vulnerables y el número de columnas, el atacante construye una consulta maliciosa utilizando `UNION SELECT`. Por ejemplo: *http://example.com/page.php?id=1 UNION SELECT username, password FROM users*. En este caso, los datos sensibles de la tabla `users` son combinados con la consulta original.

Para acceder a la sección especifica para esta inyección en el laboratorio, una vez desplegado el servidor, se