Tendencias Actuales de las Vulnerabilidades y Ataques de inyección SQL

Current Trends in SQL Injection Vulnerabilities and Attacks

Sebastian David Duque Muñoz 1[1]
sebastian.duque00@usc.edu.co

Byron Leandro Montero Muñoz 21
byron.montero00@usc.edu.co

Henry Raúl González Brito 3, M.Sc2
henryraul@uci.cu

Yaimí Trujillo Casañola 4, M.Sc3
yaimi@uci.cu

Universidad Santiago de Cali, Facultad de Ingeniería, Programa de Ingeniería de Sistemas (1)
Universidad de las Ciencias Informáticas, Departamento de Ciberseguridad, Facultad 2 (2)
Universidad de las Ciencias Informáticas, Dirección de Calidad de Software (3)

Resumen
El objetivo de la investigación fue realizar un análisis sobre las vulnerabilidades de ataques de inyección SQL, visto desde las temáticas trabajadas en investigaciones recientes y los reportes internacionales. La metodología consistió en la revisión sistemática que permite conocer el estado de actual abordada a partir de varias interrogantes: ¿Cuál es la tendencia en detecciones de ataques de iny ección SQL? ¿Cuáles son las principales técnicas para identificar este tipo de vulnerabilidad? ¿Cómo se manifiestan estos ataques? ¿Qué herramientas se emplean para identificar este tipo de vulnerabilidad? ¿Cuáles son las medidas para mitigar que se explote este tipo de vulnerabilidad? Las bases de datos utilizadas fueron IEEE, ACM, Elservier tomando como período desde 2020 al 2024. Los resultados muestran que anualmente se han reportado hasta 2000 ataques de inyección SQL, lo que demuestra a lo largo de los años un aumento significativo. Los ataques de inyección SQL se manifiestan al permitir robo de datos, manipulación de datos, acceso no autorizado, denegación de servicios entre otros. Los tipos de ataques abordados en el artículo son: básica, a ciegas, basadas en errores y basadas en UNION. Las principales técnicas para identificar estas vulnerabilidades son: análisis estático de código, pruebas de penetración, escaneo de vulnerabilidad y revisión de logs, con el empleo de las herramientas SQLMap, OWASP ZAP, Burp Suite, Acunetix, entre otras. Las principales medidas para mitigar la presencia este tipo de vulnerabilidades e impedir su explotación se orientan a: validación y procesamiento de entradas, uso de consultas parametrizadas, escaneo regular de vulnerabilidades, aplicación de actualizaciones de seguridad y auditorías de seguridad.

Palabras Clave: *amenaza; hacker ético; inyección SQL; seguridad; vulnerabilidad.*

Abstract
The objective of the research was to explore the vulnerabilities of SQL injection attacks, seen from the topics covered in recent research and international reports. The methodology consisted of a systematic review that allows us to know the current status addressed from several questions: What is the trend in detections of SQL injection attacks? What are the main techniques to identify this type of vulnerability? How do these attacks manifest? What tools are used to identify this type of vulnerability? What are the measures to mitigate this type of vulnerability from being exploited? The databases used were IEEE, ACM, Elservier, taking the period from 2020 to 2024. The results show that up to 2,000 SQL injection attacks have been reported annually, which demonstrates a significant increase over the years. SQL injection attacks manifest themselves by allowing data theft, data manipulation, unauthorized access, denial of services, among others. The types of attacks discussed in the article are: basic, blind, error-based, and UNION-based. The main techniques to identify these vulnerabilities are: static code analysis, penetration testing, vulnerability scanning and log review, using the tools

SQLMap, OWASP ZAP, Burp Suite, Acunetix, among others. The main measures to mitigate the presence of this type of vulnerabilities and prevent their exploitation are aimed at: input validation and processing, use of parameterized queries, regular vulnerability scanning, application of security updates and security audits.

Keywords: *threat; ethical hacker, SQL injection; security; vulnerability.*

1. INTRODUCCIÓN

La ciberseguridad es un componente esencial para el éxito de la transformación digital. A medida que las organizaciones trasladan sus operaciones, servicios y datos hacia plataformas digitales, la protección de sistemas, aplicaciones y bases de datos se vuelve una prioridad crítica (Möller, 2023). Las amenazas cibernéticas, como el robo de datos, la manipulación de información o la interrupción de servicios, pueden comprometer la disponibilidad, integridad y confidencialidad de los activos digitales, provocando consecuencias severas como pérdidas económicas, daños reputacionales o sanciones legales (Díaz Jiménez et al., 2023).

Entre las vulnerabilidades más comunes se encuentran los ataques de inyección, donde los adversarios aprovechan debilidades en la validación de entradas de usuario para insertar código malicioso que será ejecutado por la aplicación. Particularmente, la inyección SQL (SQLi) ha demostrado ser una de las técnicas más frecuentes y peligrosas dentro del ámbito de la ciberseguridad (Nagendran et al., 2019). Esta técnica permite a un atacante introducir instrucciones SQL en los campos de entrada de una aplicación, las cuales son interpretadas por el motor de base de datos como parte de una consulta legítima, lo que puede eludir mecanismos de autenticación y exponer información confidencial.

Los ataques de inyección SQL pueden tener impactos significativos: robo de identidad, violaciones de datos, pérdida financiera y deterioro de la confianza institucional. Aunque esta amenaza es conocida desde hace décadas, continúa vigente debido a varios factores técnicos y organizacionales: reutilización de código inseguro, escaso conocimiento sobre el manejo de entradas en diferentes lenguajes de programación, y falta de implementación de prácticas seguras de desarrollo (Hajar et al., 2024).

La persistencia de estas vulnerabilidades exige un análisis técnico riguroso. De acuerdo con Nagendran et al. (2019), la efectividad del ataque varía según el lenguaje de programación, el tipo de tecnología empleada (interpretada o compilada), y el modelo de interacción entre cliente, servidor y base de datos. Lenguajes interpretados como PHP y Python son especialmente vulnerables si el código se ejecuta sin validación en tiempo de ejecución. En contraste, lenguajes compilados como Java o .NET también pueden ser explotados si las consultas SQL se construyen mediante concatenación de cadenas, incluso si han sido compiladas previamente.

Además, la amenaza no es homogénea: su efectividad y forma de explotación varían dependiendo de cómo se gestiona la entrada del usuario, el tipo de base de datos, y el flujo de interacción entre cliente, servidor y motor SQL (Hasanov et al., 2024; Bastidas Fuertes et al., 2023). Por ello, se requiere un análisis técnico exhaustivo que no solo describa la vulnerabilidad, sino que identifique sus manifestaciones, vectores de ataque y contramedidas específicas según el stack tecnológico.

El presente artículo tiene como objetivo analizar de forma integral el fenómeno de la inyección SQL

en entornos web modernos. Para ello, se abordan los siguientes ejes:

- Las condiciones técnicas y lógicas que habilitan el ataque (validación deficiente, acceso a consultas sin control, exposición directa al motor SQL).
- El comportamiento de distintos lenguajes de programación (PHP, Python, JavaScript, Java, C#) frente a la construcción y ejecución de sentencias SQL.
- Los vectores más comunes de explotación, incluyendo formularios de autenticación, buscadores internos y URLs con parámetros GET.
- La comparación entre distintos tipos de ataques SQLi (básico, a ciegas, por errores, con UNION) y su detección automatizada mediante herramientas como WebInspect, Burp Suite y OWASP ZAP.
- Estrategias de mitigación efectivas basadas en consultas parametrizadas, validación de entradas y prácticas seguras de codificación.

Con base en literatura reciente y análisis empírico, este estudio pretende aportar una visión actualizada, práctica y técnica sobre el impacto de SQLi y los mecanismos disponibles para su prevención, contribuyendo al fortalecimiento de la seguridad en el desarrollo de aplicaciones web.

2. MATERIALES Y MÉTODOS

Para el desarrollo de la presente investigación se emplearon métodos combinados que proporcionan una base para el análisis de tendencias en vulnerabilidades y ataques de inyección SQL. Entre los métodos de trabajos científicos utilizados en la investigación se destacan los que se mencionan a continuación. Además, se brinda una breve explicación de los fines para los que fueron utilizados.

Métodos teóricos

- Histórico - lógico: Para realizar un análisis de tendencia de las vulnerabilidades y ciberataques desde el 2020 al 2024 a través de investigaciones recientes y los reportes internacionales; con el fin de evaluar la tendencia y comportamiento de esta vulnerabilidad.
- Analítico - sintético: Para analizar los conceptos de ciberseguridad, vulnerabilidades y ataques de inyección SQL como antecedentes de la investigación y resumir las principales amenazas para el desarrollo de aplicaciones Web; con el fin de identificar patrones asociados a esta vulnerabilidad, analizar avances tecnológicos para mitigar la vulnerabilidad e investigar herramientas que permitan respaldar la seguridad de sistema.

Métodos empíricos

- Revisión sistemática a la bibliografía: para reunir evidencia que demuestre la necesidad de la investigación, a partir de la revisión de las vulnerabilidades y ataques recogidos entre el 2020 y el 2024 en bases de datos científicas como IEEE Xplore, ACM Digital Library, y Google Scholar, enfocándose en publicaciones desde 2020 a 2024. Los términos de búsqueda incluyeron " SQL injection attacks", "SQL Injection vulnerabilities", " SQL injection detection techniques", "SQLI mitigation", y "SQLI tools".
- Experimentación: para detectar y reducir las vulnerabilidades de inyección SQL. Se crearon entornos de prueba para simular ataques de inyección SQL y evaluar la efectividad de diferentes técnicas de mitigación.

3. RESULTADOS Y DISCUSIÓN

A continuación, se analizan los resultados de la investigación sobre las tendencias en la vulnerabilidad de inyección SQL entre los años 2020 a 2024 donde se estudian el comportamiento de estas vulnerabilidades, las técnicas actuales en detección y mitigación de este ataque; así como las principales herramientas utilizadas para su detección y las medidas preventivas para mitigar su explotación.

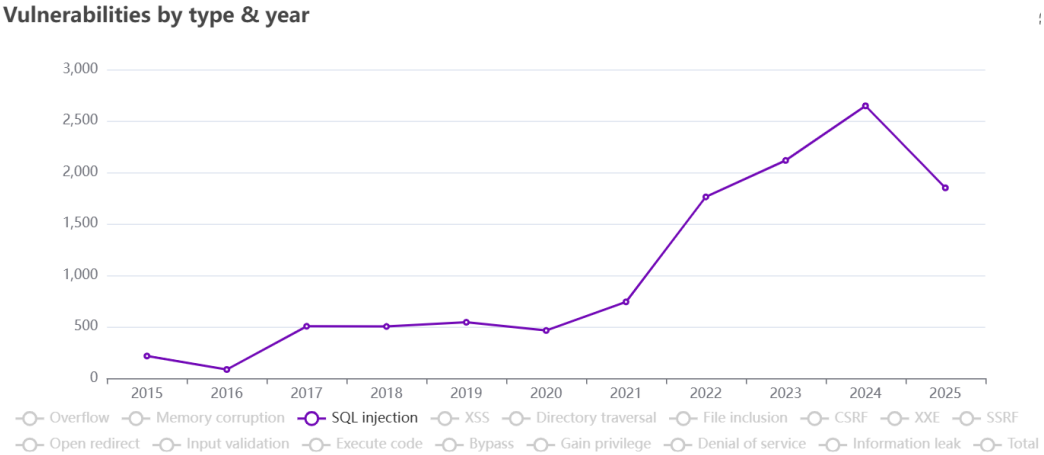
En la sección conclusión, se presentan las consecuencias que se extraen de la revisión, propuestas de nuevas hipótesis y líneas de investigación concretas para el futuro. Las conclusiones se proponen con base en las inferencias más importantes realizadas en la revisión.

Tendencias de detección de ataques de inyección SQL

La tendencia en detecciones de ciberataques de inyección SQL ha mostrado un aumento significativo entre los años 2020 a 2024. Según diversos informes de seguridad, las organizaciones perciben un incremento en la sofisticación y frecuencia de estos ataques. Los adversarios emplean técnicas avanzadas aprovechando la automatización para escanear y explotar estas vulnerabilidades en tiempo real (Arasteh et al., 2024).

En la Figura 1 se muestra el comportamiento de las vulnerabilidades de inyección SQL entre los años 2020 y 2023. Donde puede observarse como existe un incremento sostenido de esta vulnerabilidad a nivel internacional. En el año 2024 se reportan 1411 casos solo hasta el mes de junio, por ello es probable que al cierre de dicho año se mantenga esta tendencia creciente. El análisis de estos resultados indica que la existencia de vulnerabilidades de tipo SQL siguen siendo un desafío de ciberseguridad significativo. El análisis del comportamiento de estos datos puede identificar una tendencia que refleja un aumento continuo en el número de vulnerabilidades de tipo inyección SQL en los próximos años.

Figura 1. Reporte de vulnerabilidad de inyección SQL registradas del 2020 a junio 2025



Fuente: Tomado de cvedetails.com.

Manifestaciones de los Ataques de Inyección SQL

Los ataques de inyección SQL pueden manifestarse de diversas maneras dependiendo del objetivo del atacante y la naturaleza de la vulnerabilidad explotada. Según (Cumi-Guzman et al., 2024)**,** los ataques de inyección SQL se manifiestan de diversas formas, dependiendo del objetivo del atacante y la vulnerabilidad explotada. Las manifestaciones más comunes incluyen:

- **Robo de datos**: uno de los objetivos principales de los ataques de inyección SQL es obtener acceso no autorizado a datos sensibles almacenados en la base de datos. Los atacantes pueden usar inyecciones SQL para extraer información confidencial, como nombres de usuarios, contraseñas, direcciones de correo electrónico, números de tarjetas de crédito, y otros datos personales. También pueden comprometer información de clientes, como historiales de compras y datos de contacto, puede ser robada y utilizada para fraudes o ventas en mercados negros (Abdullayev & Chauhan, 2023).

- **Manipulación de datos**: los atacantes pueden alterar los datos almacenados en la base de datos, causando graves repercusiones en la integridad de la información. Las repercusiones pueden ser las modificaciones de registros con los que cambiar valores en la base de datos, como precios de productos, saldos de cuentas o niveles de permisos de usuario, eliminación de los datos o registros cruciales, lo que puede llevar a la pérdida de datos importantes y afectar el funcionamiento de la aplicación e inserción de datos falsos lo que puede desestabilizar la lógica de la aplicación y causar fallas operativas (Gandhi et al., 2021).

- **Acceso no autorizado**: los ataques de inyección SQL pueden ser utilizados para obtener privilegios elevados en el sistema afectado, estas radican por ejemplo en elevación de privilegios teniendo acceso a cuentas administrativas o acceso a funciones restringidas de la aplicación (Abdullayev & Chauhan, 2023).

- **Denegación de servicio**: aunque menos común, la inyección SQL puede ser utilizada para causar una denegación de servicio, haciendo que la base de datos y, por ende, la aplicación se vuelva inaccesibles mediante sobrecarga de la base de datos o bloqueo de recursos para impedir que otros usuarios accedan a la base de datos (Rohini et al., 2020)

Las manifestaciones de los ataques de inyección SQL son variadas y pueden tener impactos significativos en la seguridad y la integridad de los datos de una organización. Entender cómo se presentan estos ataques es crucial para desarrollar estrategias efectivas de detección y mitigación. Las organizaciones deben estar vigilantes y utilizar un enfoque proactivo para proteger sus sistemas contra estas amenazas (Cumi-Guzman et al., 2024).

Casos más comunes donde se realizan ataques SQLi

En la Figura 2 se ilustran las principales manifestaciones de los ataques de inyección SQL, evidenciando los impactos más frecuentes que pueden derivarse de este tipo de vulnerabilidad. Entre ellos se encuentran el robo de datos sensibles, la manipulación maliciosa de información, el acceso no autorizado a sistemas y recursos, y los ataques orientados a causar denegación de servicio. Estos escenarios representan riesgos críticos para la integridad, confidencialidad y disponibilidad de los datos gestionados por aplicaciones web vulnerables.

Figura 2. Ataques de inyección SQL

Fuente: Imagen generada por inteligencia artificial mediante ChatGPT, 2025

Para comprender adecuadamente las manifestaciones de los ataques de inyección SQL, es necesario abordar también los vectores de ataque más frecuentes y las técnicas empleadas por los atacantes. A continuación, se describen los escenarios más comunes donde se explotan vulnerabilidades mediante inyección SQL, como los formularios de autenticación, los campos de búsqueda en sitios web y las URL con parámetros GET. Asimismo, se detallan los diferentes tipos de ataques SQLi básicos, ciegos, basados en errores y mediante UNION junto con su ejecución técnica en diversos lenguajes de programación, destacando tanto los ejemplos vulnerables como las mejores prácticas de mitigación recomendadas.

Formularios de autenticación (Login)

Es el punto más explotado. Los atacantes inyectan comandos SQL en campos como usuario o contraseña para evadir la autenticación y obtener acceso no autorizado. (Hu et al., 2020) explican que más del 60% de los vectores iniciales en SQLi documentados entre 2016 y 2020 provienen de formularios de autenticación inseguros.

El atacante escribe lo siguiente en el campo “usuario”:

usuario: ' OR '1'='1

Y deja la contraseña vacía.

La consulta se transforma en:

SELECT * FROM users WHERE username = '' OR '1'='1' AND password = ''

'1'='1' siempre será verdadero, por lo que el sistema autoriza el acceso, aunque la contraseña no sea válida.

Búsquedas en sitios web (Search Boxes o filtros)

Los campos de búsqueda que incorporan los términos introducidos directamente en las consultas son

blanco fácil cuando no hay filtrado de entrada. (Demilie & Deriba, 2022) documentan inyecciones exitosas en campos de búsqueda de sistemas de gestión de bibliotecas, e-commerce y CMS mal configurados.

Un atacante puede ingresar en el buscador:

%' OR '1'='1

La consulta resultante sería:

SELECT * FROM productos WHERE nombre LIKE '%%' OR '1'='1%'

Esto devuelve todos los productos, sin importar la búsqueda original, y puede permitir extracción masiva de información.

URLs con parámetros GET (ID de usuario, artículo, producto)

Cuando los identificadores enviados en la URL se insertan directamente en consultas sin escape, el atacante puede manipularlos. Por ejemplo un atacante puede modificar la URL así:

http://site.com/product.php?id=5' OR 1=1--

Esto convierte la consulta en:

SELECT * FROM productos WHERE id = 5 OR 1=1—

Esto devuelve todos los productos, o incluso permite leer información confidencial si la tabla está mal protegida.

Tipos de ataques de inyección SQL

A continuación, se analizan las clasificaciones de ataques de inyección SQL, destacando sus métodos, objetivos y posibles consecuencias.

Inyección SQL básica

Es una técnica de ataque en la que se explota una vulnerabilidad en una aplicación web para inyectar código SQL malicioso en los campos de entrada de la aplicación. Este código se ejecuta en la base de datos, permitiendo manipular las consultas SQL que la aplicación envía a la base de datos.

Una aplicación web tiene una página de inicio de sesión que verifica las credenciales del usuario con una consulta SQL como esta:

SELECT * FROM usuarios WHERE usuario = '$usuario' AND contraseña =

'$contraseña'; Donde $usuario y $contraseña son variables que toman los valores

ingresados por el usuario.

Si la aplicación no valida adecuadamente los datos de entrada, un atacante podría introducir código SQL malicioso en el campo de usuario o contraseña. Por ejemplo, el atacante podría ingresar lo siguiente en el campo de usuario:

admin' or 1=1–

La consulta SQL resultante sería:

SELECT * FROM usuarios WHERE usuario = 'admin' or 1=1 --' AND contraseña = '';

Los guiones dobles (--) es un comentario en SQL, lo que significa que el resto de la consulta se ignora. Por lo tanto, la consulta se convierte en:

SELECT * FROM usuarios WHERE usuario = 'admin' or 1=1 --

El análisis devolverá todos los registros de la tabla usuarios donde el nombre de usuario es admin, sin importar la contraseña. Esto permitiría al atacante iniciar sesión como el usuario admin sin conocer la contraseña (Halfond et al., 2024).

Inyección SQL a ciegas

Las inyecciones SQL a ciegas son técnicas en la que un atacante explota una vulnerabilidad en una aplicación web que permite la inyección de código SQL, pero en lugar de obtener resultados directamente de la base de datos, debe deducir la información basándose en las respuestas de la aplicación. Esto se llama a ciegas porque el atacante no tiene acceso directo a los datos de la base de datos, sino que debe inferir la información a través de la interacción con la aplicación.

Para extraer datos, el atacante podría usar consultas como:

usuario' AND SUBSTRING((SELECT contraseña FROM usuarios WHERE usuario = 'admin'), 1, 1) = 'a' –

Verificar si el primer carácter de la contraseña del usuario 'admin' es 'a' si la página se comporta de manera diferente (por ejemplo, tarda más en cargar debido a una instrucción SLEEP inyectada), el atacante sabe que el primer carácter es 'a'. Luego, el atacante puede continuar probando caracteres hasta que haya deducido toda la contraseña (Crespo-Martínez, 2021).

Inyección SQL basadas en errores

La inyección SQL basadas en errores es una técnica en la que un atacante explota una vulnerabilidad en una aplicación web para causar fallas en la base de datos. Estas fallas pueden revelar información sobre la estructura de la base de datos o los datos mismos, permitiendo al atacante inferir información que de otra manera no sería accesible. Por ejemplo, si la aplicación no valida adecuadamente los datos de entrada, un atacante podría introducir un código SQL malicioso en el campo de usuario o contraseña para provocar la publicación de un mensaje de error.

Inyección SQL basada en UNION

Es un tipo específico de ataque de inyección SQL que utiliza el comando UNION para combinar los resultados de dos o más consultas SQL en un solo conjunto de resultados. Este tipo de ataque se utiliza para ejecutar consultas adicionales en la base de datos y obtener información sensible. La base de datos devolvería un conjunto de resultados que incluye tanto los productos como los nombres y contraseñas de los usuarios (Hu et al., 2020).

Ejecución técnica del ataque por lenguaje de programación

El impacto de un ataque de inyección SQL varía significativamente según el lenguaje de programación utilizado, su modelo de ejecución (interpretado o compilado) y el tipo de conexión que la aplicación mantiene con la base de datos. Esta sección presenta un análisis técnico de cómo se construyen y ejecutan las consultas SQL en los lenguajes más comunes en desarrollo web: PHP, Python, JavaScript/TypeScript, Java y .NET/C#, destacando las prácticas inseguras y las medidas de mitigación adecuadas para cada caso.

PHP

PHP, por su naturaleza interpretada y su histórico uso en desarrollo web, es uno de los lenguajes más frecuentemente explotados mediante inyección SQL. La concatenación directa de datos del usuario en consultas es el patrón de vulnerabilidad más común.

Como se puede observar en la Figura 3:

Figura 3. Fragmento de código en PHP vulnerable a inyección SQL

```
$user = $_POST['usuario'];
$pass = $_POST['clave'];
$sql = "SELECT * FROM users WHERE username = '$user' AND password = '$pass'";
```

Fuente: Elaboración propia 2025.

No obstante un ejemplo seguro se puede observar en la Figura 4:

Figura 4. Fragmento de código seguro en PHP vulnerable a inyección SQL

```
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = ? AND password = ?");
$stmt->execute([$user, $pass]);
```

Fuente: Elaboración propia 2025.

Esta mitigación Previene inyección SQL porque los parámetros se tratan como datos, no como código SQL (Bastidas Fuertes et al., 2023). El método prepare() compila y optimiza una consulta SQL en las bases de datos, mientras que execute() ejecuta dicha consulta previamente preparada sustituyendo los marcadores con los valores reales proporcionados.

Python

En Python, el uso de bibliotecas como sqlite3, psycopg2 o SQLAlchemy puede facilitar o prevenir inyecciones, dependiendo de si se usan parámetros seguros o interpolación directa.

Como se puede observar en la Figura 5:

Figura 5. Fragmento de código en Python vulnerable a inyección SQL

```
import sqlite3

user = request.form['username']
query = "SELECT * FROM users WHERE username = '%s'" % user

conn = sqlite3.connect('PEIPI.db')
cursor = conn.cursor()
cursor.execute(query)
```

Fuente: Elaboración propia 2025.

No obstante un ejemplo seguro se puede observar en la Figura 6:

Figura 6. Fragmento de código seguro en python vulnerable a inyección SQL

```
query = "SELECT * FROM users WHERE username = ?"
cursor.execute(query, (user,))
```

Fuente: Elaboración propia 2025.

El uso de placeholders posicionales es clave para sustituir el valor directo de la cadena SQL para evitar la ejecución arbitraria mediante este marcador de posición (Tighilt et al., 2023).

JavaScript / TypeScript (Node.js)

En el backend con Node.js, frameworks como Express y bibliotecas como mysql2, Sequelize o Prisma permiten conexiones SQL que pueden ser explotadas si se construyen incorrectamente.

Como se puede observar en la Figura 7:

Figura 7. Fragmento de código en JavaScript vulnerable a inyección SQL

```
const sql = `SELECT * FROM users WHERE username = '${user}'`;
db.query(sql, (err, res) => {});
```

Fuente: Elaboración propia 2025.

No obstante un ejemplo seguro se puede observar en la Figura 8:

Figura 8. Fragmento de código seguro en JavaScript vulnerable a inyección SQL

```
const sql = "SELECT * FROM users WHERE username = ?";
db.query(sql, [user], (err, res) => {});
```

Fuente: Elaboración propia 2025.

El uso de consultas parametrizadas en Node.js, como muestra la figura Figura 8 permite enviar los datos por separado del código SQL. Esta separación asegura que los datos del usuario no puedan alterar la estructura de la consulta, previniendo ataques de inyección SQL. (Hu et al., 2020).

Java

Java compila el código antes de ejecutarlo, pero si se utilizan objetos Statement con concatenación, el ataque SQLi puede ocurrir.

Como se puede observar en la Figura 9:

Figura 9. Fragmento de código en Java vulnerable a inyección SQL

```
String query = "SELECT * FROM users WHERE name = '" + user + "'";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

Fuente: Elaboración propia 2025.

No obstante, un ejemplo seguro se puede observar en la Figura 10:

Figura 10. Fragmento de código seguro en Java vulnerable a inyección SQL

```
String query = "SELECT * FROM users WHERE name = ?";
PreparedStatement pstmt = conn.prepareStatement(query);
pstmt.setString(1, user);
ResultSet rs = pstmt.executeQuery();
```

Fuente: Elaboración propia 2025.

El uso de PreparedStatement garantiza la separación entre la lógica de la consulta y los datos del usuario, ya que los parámetros se envían de forma independiente al motor SQL. Esto previene que los datos ingresados sean interpretados como parte del código SQL, eliminando así el riesgo de inyección SQL (Jana & Maity, 2020).

C# / .NET

En entornos Windows, el uso de SqlCommand sin parámetros es una de las prácticas más riesgosas.

Como se puede observar en la Figura 11:

Figura 11. Fragmento de código en C# vulnerable a inyección SQL

```
string query = "SELECT * FROM users WHERE user = '" + inputUser + "'";
SqlCommand cmd = new SqlCommand(query, conn);
```

Fuente: Elaboración propia 2025.

No obstante, un ejemplo seguro se puede observar en la Figura 12:

Figura 12. Fragmento de código seguro en C# vulnerable a inyección SQL

```
string query = "SELECT * FROM users WHERE user = @user";
SqlCommand cmd = new SqlCommand(query, conn);
cmd.Parameters.AddWithValue("@user", inputUser);
```

Fuente: Elaboración propia 2025.

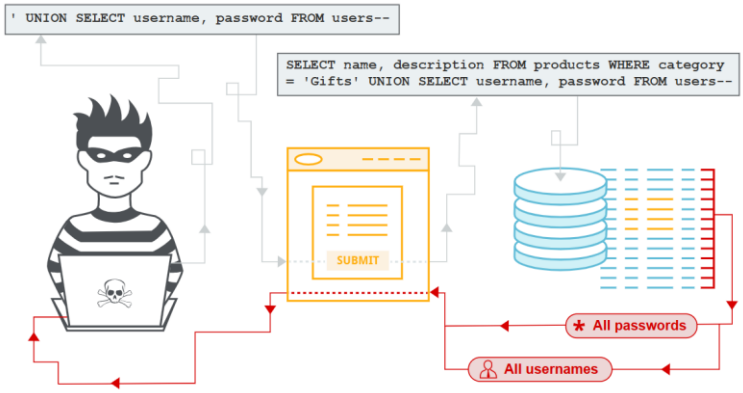
El uso de parámetros tipados en SqlCommand, como AddWithValue, permite separar el código SQL de los datos ingresados por el usuario. Esta separación evita que el input sea interpretado como parte de la consulta, bloqueando ataques de inyección SQL (Babun et al., 2021).

El ataque de inyección SQL se manifiesta de diferentes formas en función del lenguaje utilizado y su forma de ejecución. El uso de consultas parametrizadas, APIs seguras, y ORMs robustos representa la línea de defensa más efectiva en todos los entornos analizados. La concientización técnica de los desarrolladores sobre cómo se forman las consultas SQL en su stack tecnológico específico es esencial para prevenir este tipo de ataques.

Ejemplo de ataque inyección SQL

A continuación, en la Figura 13 se expone un ataque de inyección SQL en el cual se valida el tipo de ataque, lenguaje y condiciones que lo hacen posible.

Figura 13. Caso Inyección SQL



Fuente: Tomado de portswigger 2025.

Tipo de ataque

Analizando la Figura 13 el tipo de ataque de inyección SQL usado es basado en UNION. Este ataque aprovecha la cláusula UNION de SQL para combinar el resultado de dos o más consultas SELECT.

El atacante inserta una consulta maliciosa (UNION SELECT username, password FROM users) esta inyección modifica la lógica de la consulta original, combinando su propio SELECT con el resultado legítimo que se une a la consulta original del sistema permitiendo leer datos de otras tablas, como credenciales de usuarios, sin autorización.

Lenguaje de implementación

La inyección ocurre en el lenguaje SQL, pero es enviada a través de una aplicación web escrita en cualquier lenguaje del lado del servidor, como:

- PHP
- JavaScript (Node.js)
- Python (Flask/Django)
- Java (Spring)
- C# (.NET)

Específicamente, la Figura 13 no parece indicar un lenguaje en particular, pero el payload y el flujo aplican a cualquier backend que construya dinámicamente cadenas SQL sin usar parámetros seguros.

Condiciones técnicas que hacen posible el ataque

- El input del usuario no es filtrado ni parametrizado.
- La aplicación concatena directamente el input dentro de la consulta SQL.

No se utiliza ninguna técnica como:

- PreparedStatement (Java)
- Parameterized Queries (Node.js, Python)
- AddWithValue() (C#)
- ORMs con escape automático (Prisma, Sequelize, etc.)

Tal como se ilustra en la Figura 13 este tipo de ataque permite a un atacante fusionar su propia consulta maliciosa con la legítima, accediendo así a información confidencial como usuarios y contraseñas. La prevención efectiva requiere una validación estricta del input y el uso de sentencias SQL seguras.

Principales técnicas para identificar vulnerabilidades de inyección SQL

Las vulnerabilidades de inyección SQL pueden ser difíciles de detectar si no se utilizan las técnicas adecuadas. A continuación, se detallan las principales técnicas empleadas:

- **Análisis estático de código**: el análisis estático de código implica revisar el código fuente de una aplicación sin ejecutarlo. Este método permite a los desarrolladores y auditores de seguridad identificar posibles vulnerabilidades al examinar las líneas de código que interactúan con la base de datos (Jana & Maity, 2020).

- **Pruebas de penetración**: las pruebas de penetración, o pentesting, simulan ataques reales para identificar vulnerabilidades. los pentesters intentan explotar las vulnerabilidades en un entorno controlado para evaluar la seguridad de la aplicación (Hajar et al., 2024).

- **Escaneo de vulnerabilidades**: los escáneres de vulnerabilidades son herramientas automatizadas que examinan aplicaciones web en busca de posibles fallos de seguridad, incluida la inyección SQL (Zhao & Liu, 2020).

- **Revisión de logs**: analizar los registros de las bases de datos y las aplicaciones puede revelar intentos de explotación de vulnerabilidades de inyección SQL. Este método se basa en la identificación de patrones sospechosos en las consultas SQL registradas (Muslihi & Alghazzawi, 2020).

Identificar vulnerabilidades de inyección SQL es un proceso multidimensional que requiere una combinación de técnicas manuales y automatizadas. Al emplear una estrategia integral que incluye análisis estático de código, pruebas de penetración, escaneo de vulnerabilidades, revisión de logs, las organizaciones pueden detectar y mitigar estas amenazas antes de que sean explotadas (Alenezi et al., 2021).

Herramientas para identificar vulnerabilidades de inyección SQL

Detectar y mitigar vulnerabilidades de inyección SQL es una prioridad para los especialistas en ciberseguridad. Las herramientas permiten a los analistas detectar vectores de ataque antes de que sean explotados, automatizar pruebas de penetración, y auditar el comportamiento de las aplicaciones web. A continuación, se describen las herramientas más utilizadas en el sector, seleccionadas por su prevalencia en entornos empresariales, su respaldo en literatura académica y su mención frecuente en análisis profesionales (*Zenarmor*, 2024; Radware, 2023; NordPass, 2024).

Tabla 1. Herramientas destacadas de uso profesional

Herramienta	Licencia / Costo	Nivel de uso	Capacidades principales	Complejidad
SQLMap	Gratuita / Código abierto	Alta	Detección y explotación de SQLi (boolean, time, stacked queries, etc.)	Media
OWASP ZAP	Gratuita / Código abierto	Alta	Escaneo automático + proxy para pruebas manuales	Media
Burp Suite	Versión gratuita y Pro (499 USD/año)	Muy alta	Escaneo profundo + interceptación, fuzzing y extensión personalizada	Alta
Acunetix	Comercial (a partir de ~4,500 USD/año)	Alta	Escaneo automático de aplicaciones, informes detallados y pruebas de carga	Media–Alta
Netsparker	Comercial (desde 5,000 USD)	Media–Alta	Pruebas SQLi, XSS y más con verificación automática	Alta
AppScan	Comercial (HCL, desde ~12,000 USD)	Alta (sector gobierno/empresa)	Pruebas DAST + integración con pipelines DevSecOps	Alta
WebInspect	Comercial (Fortify / Micro Focus)	Alta en banca	Escaneo dinámico a nivel empresarial	Alta

Fuente: Elaboración propia.

Según lo indicado en la Tabla 1. Herramientas destacadas de uso profesional se utilizan diversas

herramientas que permiten a los especialistas de seguridad encontrar y mitigar estas vulnerabilidades antes de que puedan ser explotadas por atacantes (Rankothge et al., 2020). A continuación, se presentan las características de las herramientas mencionadas:

- **SQLMap**: es una herramienta de código abierto que automatiza la detección y explotación de vulnerabilidades de inyección SQL. Es una de las herramientas más utilizadas debido a que puede detectar automáticamente el tipo de base de datos y las posibles vulnerabilidades de inyección SQL. Permite la explotación de estas vulnerabilidades y puede obtener información de la base de datos (Bouafia et al., 2023).

- **OWASP ZAP**: una herramienta de código abierto desarrollada por la OWASP (Open Web Application Security Project). Es una de las herramientas para evaluar la seguridad de aplicaciones web, para identificar vulnerabilidades de inyección SQL. Capaz de funcionar como un proxy que intercepta y modifica el tráfico HTTP y realiza escaneos automatizados para detectar vulnerabilidades comunes en aplicaciones Web (Zukran & Siraj, 2021).

- **Burp Suite**: es una herramienta para realizar pruebas de seguridad de aplicaciones web. Desarrollada por PortSwigger, Burp Suite es utilizada por especialistas de seguridad para detectar y explotar vulnerabilidades, incluida la inyección SQL. Permite interceptar, modificar y reenviar peticiones HTTP para probar la seguridad de las aplicaciones web. Incluye escáner automatizado que puede detectar inyecciones SQL y otros tipos de vulnerabilidades (Rahalkar & Rahalkar, 2021).

- **Acunetix**: es un escáner de seguridad web que automatiza la detección de una amplia gama de vulnerabilidades, incluida la inyección SQL. Es utilizado por organizaciones para proteger sus aplicaciones web. Cuenta con un escaneo de aplicaciones web para identificar diversas vulnerabilidades de seguridad y proporciona informes sobre las vulnerabilidades encontradas ofreciendo recomendaciones para su mitigación (Bouafia et al., 2023).

- **Netsparker (actualmente Invicti)**: es una herramienta comercial de análisis dinámico (DAST) que destaca por su motor de verificación automática de vulnerabilidades denominado *Proof-Based Scanning*. Esta funcionalidad permite confirmar que una inyección SQL es explotable, minimizando falsos positivos. Netsparker también detecta XSS y otras vulnerabilidades críticas, y se integra fácilmente en entornos de CI/CD como Jenkins, Azure DevOps y GitLab CI. Es ampliamente utilizada por medianas y grandes empresas para asegurar aplicaciones web modernas (Rohit R et al., 2023).

- **AppScan**: desarrollado inicialmente por IBM y actualmente gestionado por HCL, es una solución empresarial que combina análisis estático (SAST) y dinámico (DAST) para detectar vulnerabilidades como inyecciones SQL, XSS y desbordamientos. AppScan permite una integración robusta con pipelines DevSecOps y se destaca por su adopción en entornos regulados como banca, salud y gobierno, gracias a su cumplimiento con estándares como OWASP y PCI-DSS (appscan, 2025).

- **WebInspect**: es una herramienta de escaneo dinámico avanzada desarrollada por Fortify (Micro Focus), orientada a grandes corporaciones y entornos críticos. WebInspect detecta inyecciones SQL, XPath y otros vectores, siendo capaz de analizar aplicaciones web tradicionales, modernas e incluso tecnologías heredadas. Es frecuentemente implementada en infraestructura bancaria y de defensa por su capacidad de integración con plataformas como Fortify SSC y ArcSight (Häyrynen, 2020).

El uso de herramientas para la detección de vulnerabilidades de inyección SQL es fundamental para proteger las aplicaciones web contra estos ataques. Herramientas como SQLMap, OWASP ZAP, Burp Suite, Acunetix, Netsparker, AppScan, WebInspect ofrecen diversas funcionalidades que permiten identificar estas vulnerabilidades de manera efectiva.

Análisis comparativo

Las empresas deben adoptar una estrategia de defensa en profundidad, integrando herramientas de análisis estático (SAST), dinámico (DAST) y monitoreo en tiempo real (SIEM + WAF). Por ejemplo

Más utilizadas en empresas: *Burp Suite Pro*, *Acunetix*, y *SQLMap* (como complemento de herramientas comerciales) para pentesting en entornos de desarrollo y pruebas.

Mejores para iniciarse: *OWASP ZAP* y *SQLMap* por ser gratuitas y ampliamente documentadas.

Recomendadas para grandes organizaciones: *AppScan* y *WebInspect*, por sus capacidades de integración con DevOps y cumplimiento normativo, indicado para auditorías formales en producción.

Las herramientas para detectar inyecciones SQL cumplen un rol clave en la protección de aplicaciones web. Desde soluciones gratuitas como SQLMap y OWASP ZAP, útiles para pruebas iniciales, hasta plataformas empresariales como AppScan y WebInspect, orientadas a entornos críticos, cada una responde a necesidades específicas.

Su correcta integración en el ciclo de desarrollo seguro permite identificar vulnerabilidades antes de que sean explotadas, reduciendo riesgos y fortaleciendo la seguridad general de los sistemas. Elegir la herramienta adecuada según el entorno y nivel de madurez organizacional es esencial para una defensa eficaz frente a ataques SQLi.

Medidas para mitigar la explotación de vulnerabilidades de inyección SQL

Para mitigar la presencia de vulnerabilidades de tipo SQL es necesario aplicar un conjunto de medidas en función del tipo de la vulnerabilidad que pueda estar presente (Jana & Maity, 2020). Las principales medidas son:

- **Validación y procesamiento de entradas**: es necesario procesar y validar todas las entradas del usuario para evitar la inyección de código malicioso.
- **Uso de consultas parametrizadas**: deben emplearse consultas parametrizadas para segmentar el código SQL de los datos introducidos por el usuario y utilizar las funcionalidades de filtrado de entradas implícitas de los frameworks de desarrollo.
- **Escaneo regular de vulnerabilidades**: es importante realizar escaneos periódicos para detectar y corregir vulnerabilidades antes de que sean explotadas por los adversarios.
- **Aplicación de actualizaciones de seguridad**: la actualización y mantenimiento constante de los componentes de software, tanto a nivel de bases de datos como de aplicación, es fundamental para garantizar la protección contra inyección SQL.
- **Auditorías de seguridad**: deben realizarse auditorías de seguridad periódicas para evaluar la

efectividad de las medidas de protección implementadas.

A continuación, se relacionan los diferentes tipos de ataques de inyección SQL en la Tabla 2. Tipos de inyección SQL con sus respectivas estrategias de mitigación.

Tabla 2. Tipos de inyección SQL

Tipo de ataque	Descripción	Ejemplo	Estrategias de Mitigación
Inyección SQL básica	Inserción directa de código SQL malicioso en las consultas SQL.	SELECT * FROM usuarios WHERE usuario = 'admin' -- ' AND contraseña = 'password';	Consultas parametrizadas, validación de entradas
Inyección SQL a ciegas	Explota las respuestas verdaderas/falsas en lugar de mensajes de error.	SELECT * FROM productos WHERE id = 1 AND '1'='1';	Consultas parametrizadas, manejo adecuado de errores
Inyección SQL basada en errores	Aprovecha los mensajes de error para obtener información sobre la base de datos.	SELECT * FROM usuarios WHERE id = 1 AND (SELECT COUNT(*) FROM usuarios) = 1;	Ocultar mensajes de error detallados, validación de entradas
Inyección SQL basada en UNION	Utiliza la cláusula UNION para combinar resultados de múltiples consultas.	SELECT nombre, contraseña FROM usuarios UNION SELECT nombre, contraseña FROM admin;	Consultas parametrizadas, validación de entradas

Fuente: Elaboración propia.

Como puede observarse en la Tabla 1, es necesario aplicar medidas para garantizar una seguridad razonable ante los ataques de inyección SQL. El alcance de estas medidas, así como su materialización en el código fuente de las aplicaciones, estará condicionado por la dimensión de la base de datos, así como del número de entradas que se deban procesar. Es importante apoyarse en los mecanismos de consultas parametrizadas que brindan los frameworks de desarrollo actuales.

4. CONCLUSIONES

La presente investigación confirma que los ataques de inyección SQL continúan representando una amenaza crítica y persistente para la seguridad de las aplicaciones web. A pesar de su antigüedad como vector de ataque, su prevalencia se mantiene elevada debido a prácticas inseguras de desarrollo, la falta de validación de entradas y la incorrecta implementación de mecanismos de defensa en los sistemas actuales.

La información recopilada en la revisión a la bibliografía sirve como base para futuras investigaciones sobre estos ataques y se llegan a las siguientes conclusiones:

- Existen diversas formas de ataque SQLi, siendo las más comunes: la inyección básica (que altera directamente las consultas), la inyección a ciegas (que infiere información sin respuestas directas), las inyecciones basadas en errores (que aprovechan mensajes del sistema) y las inyecciones mediante el operador UNION (que permiten combinar resultados de múltiples consultas).

- Las técnicas de detección más utilizadas incluyen el análisis estático de código fuente, pruebas de penetración, escaneo automatizado de vulnerabilidades y revisión de registros de sistema. Herramientas como SQLMap, Burp Suite, OWASP ZAP, WebInspect y Acunetix han demostrado ser eficaces en la identificación de estas amenazas, tanto en fases de desarrollo como en producción.
- Las estrategias de mitigación más efectivas se centran en la validación rigurosa de entradas de usuario, el uso exclusivo de consultas parametrizadas o preparadas, el escaneo regular de vulnerabilidades, la aplicación de actualizaciones de seguridad, y la ejecución de auditorías periódicas del código y la arquitectura del sistema.

Pese a la existencia de marcos de trabajo, herramientas avanzadas y guías de buenas prácticas ampliamente documentadas, muchas organizaciones siguen omitiendo su implementación sistemática. Esta brecha entre el conocimiento técnico disponible y su aplicación efectiva en entornos reales contribuye a la perpetuación de vulnerabilidades críticas.

Por ello, es imperativo que tanto desarrolladores como responsables de seguridad adopten un enfoque preventivo y proactivo, integrando la seguridad como parte intrínseca del ciclo de vida del software. La concientización técnica, el uso responsable de herramientas de análisis y la adopción de estándares como OWASP deben consolidarse como prácticas fundamentales para reducir la superficie de ataque y fortalecer la resiliencia de las aplicaciones frente a amenazas de inyección SQL.

5. REFERENCIAS

Abdullayev, V., & Chauhan, A. S. (2023). SQL injection attack: Quick view. *Mesopotamian Journal of CyberSecurity*, *2023*, 30-34.

Alenezi, M., Nadeem, M., & Asif, R. (2021). SQL injection attacks countermeasures assessments. *Indonesian Journal of Electrical Engineering and Computer Science*, *21*(2), 1121-1131.

Arasteh, B., Aghaei, B., Farzad, B., Arasteh, K., Kiani, F., & Torkamanian-Afshar, M. (2024). Detecting SQL injection attacks by binary gray wolf optimizer and machine learning algorithms. *Neural Computing and Applications*, *36*(12), 6771-6792. https://doi.org/10.1007/s00521-024-09429z

Bouafia, R., Benbrahim, H., & Amine, A. (2023). Automatic Protection of Web Applications Against SQL Injections: An Approach Based On Acunetix, Burp Suite and SQLMAP. *2023 9th International Conference on Optimization and Applications (ICOA)*, 1-6.

Crespo-Martínez, E. (2021). Análisis de vulnerabilidades con sqlmap aplicada a entornos APEX 5. *Ingenius. Revista de Ciencia y Tecnología*, *25*, 104-113. https://doi.org/10.17163/ings.n25.2021.10

Cumi-Guzman, B. A., Espinosa-Chim, A. D., Orozco-del-Castillo, M. G., & Recio-García, J. A. (2024). *Counterfactual Explanation of a Classification Model for Detecting SQL Injection Attacks*.

Diaz Jimenez, C. D., Ariza Rodriguez, E., & Ruiz Moncada, M. Y. (2023). *La Ciberseguridad en las Pymes* [Bachelor Thesis, Universidad EAN]. https://repository.universidadean.edu.co/handle/10882/12818

Gandhi, N., Patel, J., Sisodiya, R., Doshi, N., & Mishra, S. (2021). A CNN-BiLSTM based approach for detection of SQL injection attacks. *2021 International conference on computational intelligence and knowledge economy (ICCIKE)*, 378-383.

Hajar, S., Jaafar, A. G., & Rahim, F. A. (2024). A Review of Penetration Testing Process For Sql Injection Attack. *Open International Journal of Informatics*, *12*(1), Article 1. https://doi.org/10.11113/oiji2023.11n2.256

Halfond, W. G. J., Viegas, J., & Orso, A. (2024). *A Classification of SQL Injection Attacks and Countermeasures.*

Hu, J., Zhao, W., & Cui, Y. (2020). A Survey on SQL Injection Attacks, Detection and Prevention. *Proceedings of the 2020 12th International Conference on Machine Learning and Computing*, 483-488. https://doi.org/10.1145/3383972.3384028

Jana, A., & Maity, D. (2020). Code-based analysis approach to detect and prevent SQL injection attacks. *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1-6.

Muslihi, M. T., & Alghazzawi, D. (2020). Detecting SQL injection on web application using deep learning techniques: A systematic literature review. *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*, 1-6.

Nagendran, K., Adithyan, A., Chethana, R., Camillus, P., & Varshini, K. B. S. (2019). Web application penetration testing. *Int. J. Innov. Technol. Explor. Eng*, *8*(10), 1029-1035.

Rahalkar, S., & Rahalkar, S. (2021). Introduction to Burp Suite. *A Complete Guide to Burp Suite: Learn to Detect Application Vulnerabilities*, 1-10.

Rankothge, W., Randeniya, M., & Samaranayaka, V. (2020). Identification and mitigation tool for Sql injection attacks (SQLIA). *2020 IEEE 15th International Conference on Industrial and Information Systems (ICIIS)*, 591-595.

Rohini, K., Kasturi, K., & Vignesh, R. (2020). Method for simulating SQL injection and DOS attack. *Intelligent Computing and Innovation on Data Science: Proceedings of ICTIDS 2019*, 793-801.

Zhao, J., & Liu, C. (2020). Design and implementation of SQL injection vulnerability scanning tool. *Journal of Physics: Conference Series*, *1575*(1), 012094.

Zukran, B., & Siraj, M. M. (2021). Performance comparison on sql injection and xss detection using open

source vulnerability scanners. *2021 International Conference on Data Science and Its Applications (ICoDSA)*, 61-65.

appscan. (2025). *Escaneo de aplicaciones*. https://cloud.appscan.com/

Babun, L., Denney, K., Celik, Z. B., McDaniel, P., & Uluagac, A. S. (2021). A survey on IoT platforms: Communication, security, and privacy perspectives. *Computer Networks*, *192*, 108040. https://doi.org/10.1016/j.comnet.2021.108040

Bastidas Fuertes, A., Pérez, M., & Meza, J. (2023). Transpiler-Based Architecture Design Model for Back-End Layers in Software Development. *Applied Sciences*, *13*(20), Article 20. https://doi.org/10.3390/app132011371

ChatGPT. (2025). *Manifestaciones de los ataques de inyección SQL [Imagen generada por IA]. ChatGPT*. https://chat.openai.com

Demilie, W. B., & Deriba, F. G. (2022). Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques. *Journal of Big Data*, *9*(1), 124. https://doi.org/10.1186/s40537-022-00678-0

Hasanov, I., Virtanen, S., Hakkala, A., & Isoaho, J. (2024). Application of Large Language Models in Cybersecurity: A Systematic Literature Review. *IEEE Access*, *12*, 176751-176778. https://doi.org/10.1109/ACCESS.2024.3505983

Häyrynen, E. (2020). *Evaluation of state-of-the-art web application vulnerability scanners*. https://lutpub.lut.fi/handle/10024/161014

Hu, J., Zhao, W., & Cui, Y. (2020). A Survey on SQL Injection Attacks, Detection and Prevention. *Proceedings of the 2020 12th International Conference on Machine Learning and Computing*, 483-488. https://doi.org/10.1145/3383972.3384028

Jana, A., & Maity, D. (2020). Code-based Analysis Approach to Detect and Prevent SQL Injection Attacks. *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 1-6. https://doi.org/10.1109/ICCCNT49239.2020.9225575

NordPass. (2024). *Qué es una inyección de SQL*. NordPass. https://nordpass.com/es/blog/what-is-sql-injection/

Radware. (2023). *SQL Injection: Examples, Real Life Attacks & 9 Defensive Measures | Radware*. https://www.radware.com/cyberpedia/application-security/sql-injection/

Rohit R, I, H. F., M, A., J, A. M., & S, D. (2023). Web Application Security Testing Framework using Flask. *2023 2nd International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, 1646-1652. https://doi.org/10.1109/ICAAIC56838.2023.10140422

Tighilt, R., Abdellatif, M., Trabelsi, I., Madern, L., Moha, N., & Guéhéneuc, Y.-G. (2023). On the maintenance support for microservice-based systems through the specification and the detection of microservice antipatterns. *Journal of Systems and Software*, *204*, 111755. https://doi.org/10.1016/j.jss.2023.111755

Zenarmor. (2024, abril 28). https://www.zenarmor.com/docs/network-security-tutorials/best-sql-injection-detection-tools