Corporación Universitaria Reformada

Desarrollo De Una Aplicación Web Para La Detección Automatizada De Vulnerabilidades De Inyección SQL, Con Generación De Reportes Y Simulación De Ataques En Entornos Controlados

Autor(es):

Valentina Espinoza Ríos

Sebastián Pérez Quiroga

Trabajo de grado como prerrequisito para la obtención de grado de Ingeniero Informático

Director(a)

Hernando Domínguez Gutiérrez

Facultad De Ingenierías

Programa De Ingeniería Informática

Colombia – Barranquilla

2025

Corporación Universitaria Reformada

Desarrollo De Una Aplicación Web Para La Detección Automatizada De Vulnerabilidades De Inyección SQL, Con Generación De Reportes Y Simulación De Ataques En Entornos Controlados

Autor(es):

Valentina Espinoza Ríos

Sebastián Pérez Quiroga

Facultad De Ingenierías

Programa De Ingeniería Informática

Colombia – Barranquilla

2025

Índice

TABLA DE FIGURAS

Resumen

La seguridad de las aplicaciones web representa uno de los mayores desafíos de la era digital, siendo la inyección SQL (SQL Injection) una de las vulnerabilidades más críticas según el OWASP Top 10, debido a su capacidad de comprometer la confidencialidad, integridad y disponibilidad de la información. Ante la necesidad de herramientas que combinen la detección automatizada, la simulación de ataques y la generación de informes, el presente proyecto propone el diseño e implementación de una aplicación web que integra un backend en Python y Flask, un frontend en React y SQLMap como motor de análisis, permitiendo identificar vulnerabilidades SQL en entornos controlados de forma ética y responsable.

El sistema fue desarrollado bajo la metodología ágil SCRUM y validado en un entorno local con vulnerabilidades intencionales, alcanzando un nivel de madurez tecnológica TRL 4. Los resultados evidenciaron su capacidad para detectar vulnerabilidades, clasificar niveles de riesgo y generar informes estructurados, mostrando un desempeño comparable al de herramientas como OWASP ZAP y Acunetix.

El proyecto se alinea con la Ley 1273 de 2009, la Ley 1581 de 2012 y estándares internacionales como ISO/IEC 27001:2022 y OWASP Top 10, lo que refuerza su pertinencia normativa y académica.

Palabras clave: inyección SQL; ciberseguridad; aplicaciones web; detección de vulnerabilidades; SQLMap; pruebas de penetración; seguridad informática; automatización; análisis de riesgos; OWAS

Abstract

Web application security represents one of the major challenges of the digital era, with SQL Injection being one of the most critical vulnerabilities according to the OWASP Top 10, due to its ability to compromise the confidentiality, integrity, and availability of information. In response to the need for tools that combine automated detection, attack simulation, and report generation, this project proposes the design and implementation of a web application that integrates a Python and Flask backend, a React frontend, and SQLMap as the analysis engine, enabling the identification of SQL vulnerabilities in controlled environments in an ethical and responsible manner.

The system was developed using the agile SCRUM methodology and validated in a local environment with intentional vulnerabilities, reaching a Technology Readiness Level (TRL) of 4. The results demonstrated its ability to detect vulnerabilities, classify risk levels, and generate structured reports, showing performance comparable to tools such as OWASP ZAP and Acunetix.

The project aligns with Law 1273 of 2009, Law 1581 of 2012, and international standards such as ISO/IEC 27001:2022 and the OWASP Top 10, reinforcing its regulatory and academic relevance.

Keywords: SQL Injection; cybersecurity; web applications; vulnerability detection; SQLMap; penetration testing; information security; automation; risk analysis; OWASP

1. Introducción

La seguridad de las aplicaciones web es un pilar fundamental en la era digital, donde el manejo de grandes volúmenes de datos sensibles y transacciones críticas es la norma. Sin embargo, esta dependencia tecnológica trae consigo amenazas persistentes y de alto impacto. Dentro de este panorama, la inyección SQL se destaca como una de las vulnerabilidades más críticas. Organizaciones como la Open Web Application Security Project (OWASP) la catalogan consistentemente en su Top 10 de riesgos, debido a la facilidad con que puede comprometer la confidencialidad, integridad y disponibilidad de la información.

A pesar de los avances en buenas prácticas de desarrollo seguro, la validación deficiente de entradas, el uso de consultas dinámicas sin parametrizar y la persistencia de sistemas heredados continúan exponiendo a miles de aplicaciones a riesgos significativos. La creciente digitalización de servicios, sumada al rápido aumento en la complejidad de las infraestructuras tecnológicas, ha ampliado la superficie de ataque para ciberdelincuentes que aprovechan estas brechas para obtener acceso no autorizado a bases de datos, manipular información o causar indisponibilidad en los sistemas.

Ante este escenario, se vuelve indispensable desarrollar herramientas que permitan detectar de forma temprana y precisa posibles vulnerabilidades antes de que sean explotadas. Sin embargo, muchas organizaciones carecen de soluciones automatizadas que proporcionen análisis en tiempo real, clasificación de riesgos y reportes estructurados que faciliten la toma de decisiones. Esto evidencia la necesidad de tecnologías accesibles y orientadas a

entornos de prueba que permitan fortalecer la seguridad desde las primeras etapas del ciclo de desarrollo.

En este contexto, el presente proyecto propone el diseño e implementación de un sistema automatizado capaz de analizar páginas web vulnerables, identificar posibles fallas de inyección SQL y generar reportes organizados que permitan evaluar el estado de seguridad de un sitio. Apoyado en SQLMap como motor principal de análisis, y complementado con una arquitectura moderna basada en Python, Flask y una interfaz web en React, el sistema busca ofrecer una herramienta práctica, eficaz y adaptable para entornos académicos y profesionales.

El proyecto se sustenta no solo en la relevancia técnica de prevenir ataques de inyección SQL, sino también en la necesidad de fortalecer las capacidades de análisis, auditoría y toma de decisiones en materia de ciberseguridad. Al proporcionar un entorno seguro, controlado y automatizado para la detección de vulnerabilidades, esta propuesta contribuye al desarrollo de soluciones tecnológicas alineadas con las políticas nacionales e internacionales de seguridad digital y al fortalecimiento de las prácticas de protección de la información en el ecosistema digital actual

2. Planteamiento del Problema

En la actualidad, muchas aplicaciones web manejan grandes volúmenes de datos sensibles y dependen de consultas SQL para su funcionamiento. Sin embargo, estas consultas pueden convertirse en un punto de entrada para atacantes cuando no están correctamente validadas. La Open Web Application Security Project (OWASP), una fundación internacional sin ánimo de lucro dedicada a mejorar la seguridad del software identifica la inyección SQL como uno de los riesgos más importantes dentro de su listado de vulnerabilidades, ubicándola en el tercer lugar de su Top 10 de vulnerabilidades, debido a su alta frecuencia y al impacto que genera en la confidencialidad, integridad y disponibilidad de los datos (OWASP, 2025). Esta problemática se origina principalmente en la validación deficiente de entradas del usuario, el uso de consultas dinámicas sin parametrizar, la falta de adopción de prácticas de desarrollo seguro y la persistencia de sistemas heredados que no incorporan mecanismos modernos de protección. Tales causas se agravaron con la creciente digitalización de servicios y el aumento en la complejidad de las infraestructuras tecnológicas, lo que ha ampliado la superficie de ataque y facilitado la explotación de esta vulnerabilidad por parte de ciberdelincuentes. (wiz, s.f.) (tecnology, s.f.) (Universidad Central del Sur (CHINA) POBOX 410083, 2010)

Aunque diversos países han desarrollado iniciativas como las Estrategias Nacionales de Ciberseguridad y Ciberdefensa, así como entidades especializadas para la gestión de incidentes y la investigación de delitos informáticos, persisten desafíos significativos relacionados con la formación de talento humano, la sensibilización de la ciudadanía y el fortalecimiento de la infraestructura tecnológica. En este contexto internacional destacan organismos clave como el Centro Cibernético Policial y el CSIRT-GOB (Computer

Security Incident Response Team – Gobierno) en Colombia; la CISA (Cybersecurity and Infrastructure Security Agency) y la Cyber Division del FBI (Federal Bureau of Investigation) en Estados Unidos; el NCSC (National Cyber Security Centre) y la NCA (National Crime Agency) en el Reino Unido; y el INCIBE (Instituto Nacional de Ciberseguridad) junto al CCN-CERT (Centro Criptológico Nacional - Computer Emergency Response Team) en España. (Tecnología, 2021)A pesar de los avances institucionales evidenciados en estas naciones, la educación continua, el fortalecimiento de las capacidades técnicas y la articulación efectiva entre los sectores público y privado siguen constituyendo factores determinantes para elevar los niveles de ciberseguridad y mitigar los riesgos emergentes. (repository.udistrital, s.f.) (Colombia, s.f.) (Agency, s.f.) (incibe, s.f.) (WikipediA, s.f.) (techradar, s.f.)

2.1 Formulación del Problema

¿Cómo diseñar e implementar un sistema eficiente y accesible que permita detectar ataques de inyección SQL, con el propósito de fortalecer la integridad y la confidencialidad de la información en aplicaciones web, sin vulnerar sistemas ni comprometer su funcionamiento?

3. Justificación

La seguridad en las aplicaciones web se ha consolidado como un componente esencial para garantizar la integridad, confidencialidad y disponibilidad de la información tanto para las organizaciones como para los usuarios finales. En este contexto, las vulnerabilidades por inyección SQL continúan siendo una de las amenazas más relevantes debido a su alta frecuencia, facilidad de explotación y alcance potencial sobre los sistema de información. Frente a esta problemática, se hace indispensable el desarrollo de herramientas eficaces, accesibles y preventivas que faciliten la detección y mitigación temprana de este tipo de ataques, contribuyendo así al fortalecimiento de la seguridad y la confianza en los entornos digitales.

Bajo esta perspectiva, el presente proyecto propone el diseño e implementación de un sistema que realice pruebas de penetración orientadas a la detección y explotación controlada de vulnerabilidades de inyección SQL en aplicaciones web. Este sistema automatiza los procesos de análisis de seguridad, permitiendo identificar tempranamente fallos que puedan comprometer la integridad o confidencialidad de los datos. Asimismo, se concibe como una herramienta educativa y de apoyo para desarrolladores, analistas de seguridad y equipos de respuesta a incidentes, fomentando la adopción de prácticas seguras dentro del ciclo de desarrollo de software.

En cuanto al impacto práctico, la herramienta beneficiará directamente a organizaciones que busquen fortalecer sus procesos de auditoría y pruebas de seguridad, a equipos técnicos encargados de la protección de infraestructura web y a usuarios finales, quienes se verán favorecidos por aplicaciones más seguras y confiables. Al facilitar la identificación de vulnerabilidades de manera automatizada, el sistema contribuye a reducir tiempos de revisión, mejorar la toma de decisiones y disminuir el riesgo asociado a ataques de inyección SQL en entornos productivos.

Desde una perspectiva académica, este trabajo aporta a un campo donde, a pesar de

existir numerosas herramientas de auditoría, aún existe un vacío relacionado con soluciones que integren análisis automatizado, simulación controlada de ataques y generación de reportes orientados tanto a la investigación como a la formación en ciberseguridad. La propuesta se integra dentro de este vacío al ofrecer un enfoque pedagógico y técnico que permite profundizar en la comprensión de los mecanismos de explotación y defensa frente a inyecciones SQL.

Finalmente, el alcance del proyecto se centra exclusivamente en vulnerabilidades de inyección SQL en aplicaciones web y en su detección mediante pruebas de penetración controladas. No se abordarán otros tipos de ataques, como inyección de comandos, vulnerabilidades de configuración o fallos de autenticación. Asimismo, el sistema está orientado exclusivamente a entornos de prueba y no a la explotación de sistemas reales, garantizando el uso ético y responsable de la herramienta.

4. Objetivos

4.1 Objetivo General

Implementar un sistema eficiente y accesible para la detección de ataques de inyección SQL, empleando SQLMap para generar reportes detallados con recomendaciones y simular ataques controlados en entornos seguros.

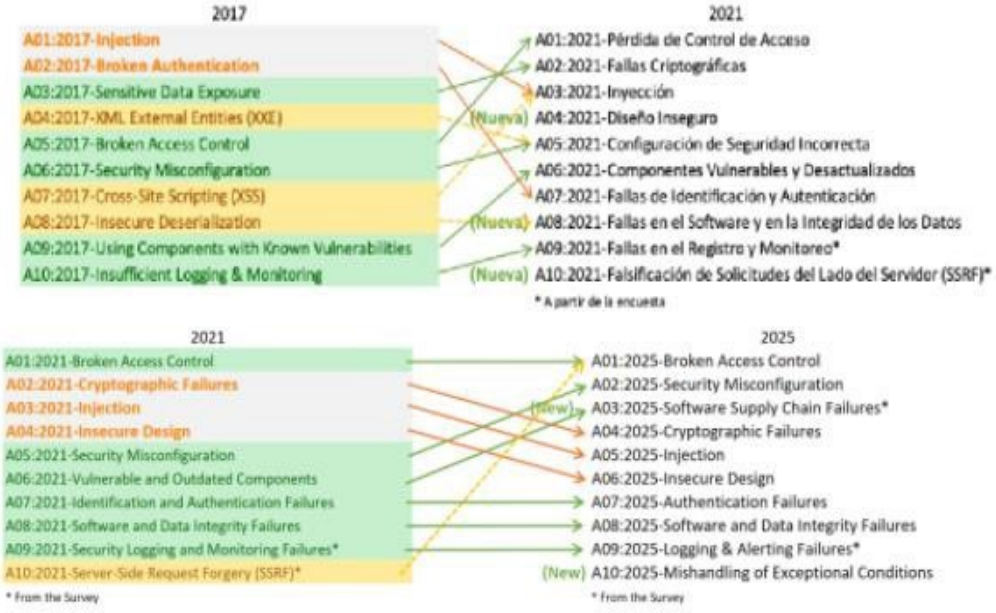
4.2 Objetivos Específicos

1. Diseñar un sistema automatizado que utilice SQLMap para detectar vulnerabilidades de inyección SQL en páginas web, incluyendo un módulo de reportes con análisis de riesgos y recomendaciones de mitigación.
2. Simular ataques controlados en entornos seguros para evaluar el impacto real de las vulnerabilidades detectadas, fortaleciendo la comprensión de los riesgos sin comprometer la seguridad de sistemas reales.
3. Validar la efectividad del sistema mediante pruebas en entornos controlados y compararlo con herramientas como OWASP ZAP, SQLMap y Acunetix para medir su precisión y eficiencia.

5. Marco Teórico

5.1 Introducción al Ataque de Inyección SQL (ISQL)

La inyección SQL (SQL Injection) es uno de los ataques más comunes y peligrosos en páginas web. Este tipo de vulnerabilidad se presenta cuando una aplicación permite que el atacante inserte (o "inyecte") comandos SQL maliciosos en una consulta hacia la base de datos de la aplicación. Este ataque puede permitir al atacante ejecutar comandos arbitrarios, lo que potencialmente da acceso no autorizado a los datos almacenados, manipulación de información, eliminación de registros, o incluso tomar control del servidor de bases de datos. Debido a su impacto severo, la inyección SQL ocupa uno de los primeros lugares en el OWASP Top 10, que es un listado anual de las vulnerabilidades más críticas en aplicaciones web según la Open Web Application Security Project (OWASP) (OWASP, 2025).



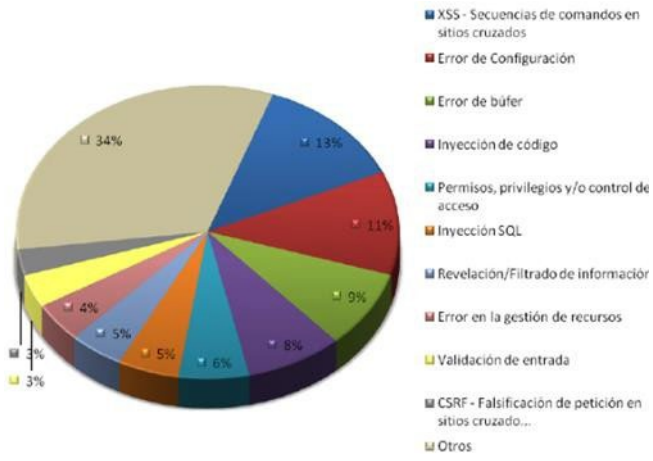
Las vulnerabilidades más comunes a nivel mundial (OWASP, 2025)

Ilustración 1

A03:2021-Inyección desciende al tercer puesto. El 94 % de las aplicaciones se analizaron para detectar algún tipo de inyección, y los 33 CWE asignados a esta categoría ocupan el segundo lugar en cuanto a incidencias. Secuencias de comandos entre sitios (CSE) ahora forman parte de esta categoría en esta edición

Para el año 2025

La inyección SQL baja dos puestos, del n.° 3 al n.° 5, en la clasificación, manteniendo su posición relativa a fallos criptográficos y diseño inseguro. La inyección de vulnerabilidades es una de las categorías más analizadas, con el mayor número de CVE asociadas a las 38 CWE de esta categoría. Incluye una variedad de problemas, desde Cross-site Scripting (alta frecuencia/bajo impacto) hasta vulnerabilidades de inyección SQL (baja frecuencia/alto impacto). (OWASP, 2025) (verizon, s.f.)



Vulnerabilidades más comunes (welivesecurity, 2012)

Ilustracion 2

Este estudio demuestra que las vulnerabilidades de Cross Site Scripting (XSS) es una de las más presentes. Asimismo, las siguientes vulnerabilidades que más impacto tuvieron son aquellas generadas por errores de configuración. Si bien este tipo de falla no

radica en el código de la aplicación o está relacionada con errores en el diseño, muchas veces las configuraciones no se realizan de forma adecuada dejando brechas de seguridad. Los errores de inyección de código son bastantes comunes también, y muchas veces se utilizan como método para alojar malware, por ejemplo, en sitios web. Finalmente, otro dato que vale la pena mencionar de este estudio, son las inyecciones SQL. Si bien sólo figura con un 5% del total de las vulnerabilidades más comunes, este tipo de ataques son muy comunes en servidores web. Estos pueden ser combinados con inyección de código o utilizados en relación a errores de configuración, por lo que el impacto que estos ataques tienen sobre la red es notable.

A continuación, se presentará un caso de la vida real donde víctimas que fueron atacadas por la vulnerabilidad de inyección SQL

TalkTalk es un proveedor de servicios de Internet en el Reino Unido. En octubre de 2015, la empresa sufrió un grave ataque cibernético que comprometió los datos personales de más de 150,000 clientes, incluyendo información financiera sensible de aproximadamente 15,000 personas. El ataque se realizó mediante una técnica conocida como *SQL injection*, un método de explotación de bases de datos que es ampliamente conocido y prevenible.

El fallo de seguridad se produjo en parte por la infraestructura heredada de la adquisición de Tiscali en 2009, páginas web vulnerables que no habían sido detectadas y software de base de datos desactualizado y sin soporte. Además, TalkTalk había recibido advertencias tempranas de ataques previos en julio y septiembre de 2015 que explotaban la misma vulnerabilidad, pero no tomó medidas adecuadas.

Como resultado de estas fallas, la Oficina del Comisionado de Información (ICO) del Reino Unido multó a TalkTalk con £400,000, cerca del máximo permitido de £500,000, calificando el incidente como una violación grave de las obligaciones de protección de datos. La ICO destacó que la empresa no había implementado medidas básicas de seguridad que podrían haber prevenido el ataque.

Un joven de 19 años, Daniel Kelley, fue acusado de realizar el ataque y de exigir un rescate en bitcoins, valorado en más de £200,000 en ese momento. TalkTalk defendió su transparencia con los clientes tras el ataque, aunque reconoció que la situación fue decepcionante.

En resumen, el incidente expuso graves deficiencias en la ciberseguridad de TalkTalk y puso en riesgo la información personal de decenas de miles de clientes, lo que resultó en una sanción histórica por parte de las autoridades regulatorias. (theguardian, s.f.)

A partir del caso de TalkTalk, donde una vulnerabilidad permitió a atacantes acceder con facilidad a información confidencial mediante SQL injection, presentaremos una descripción de cómo se llevan a cabo este tipo de ataques y por qué ocurren en aplicaciones web. Este tipo de fallos suelen aparecer cuando las aplicaciones no implementan controles básicos de validación y manejo seguro de datos.

A continuación, se describirá porque una aplicación puede ser vulnerable a ataques como SQL inyeccion

5.1 Falta de validación y filtrado: Los datos proporcionados por el usuario no son validados, filtrados ni desinfectados por la aplicación antes de ser procesados.

5.2 Consultas dinámicas inseguras: Se utilizan consultas dinámicas o llamadas no parametrizadas directamente en el intérprete, sin aplicar un escape adecuado según el contexto.

5.3 Uso inseguro en ORM: Datos maliciosos se integran en parámetros de búsqueda dentro de frameworks ORM, lo que puede permitir la extracción de registros confidenciales adicionales.

5.4 Concatenación directa de datos: Los datos hostiles se usan directamente o se concatenan al SQL o a comandos, mezclando estructura y contenido malicioso dentro de consultas dinámicas, comandos o procedimientos almacenados. (OWASP, 2025) (Tecnología, 2021)

5.5 Cómo prevenir ataques de inyección SQL

Para evitar la inyección SQL, es fundamental garantizar que los datos proporcionados por el usuario nunca se mezclen directamente con los comandos o consultas ejecutados por la aplicación. Algunas prácticas recomendadas incluyen: **(wnumeration, s.f.)**

- La opción preferida es usar una API segura que evite el uso del intérprete por completo, proporcione una interfaz parametrizada o migre a herramientas de mapeo relacional de objetos (ORM). Nota: Incluso parametrizados, los procedimientos almacenados pueden introducir una inyección SQL si PL/SQL o T-SQL concatenan consultas y datos o ejecutan datos hostiles con EXECUTE IMMEDIATE o exec().
- Utilice la validación de entrada positiva del lado del servidor. Esto no constituye una defensa completa, ya que muchas aplicaciones requieren caracteres especiales, como áreas de texto o API para aplicaciones móviles.

- Para cualquier consulta dinámica residual, escape los caracteres especiales utilizando la sintaxis de escape específica de ese intérprete. Nota: Las estructuras SQL, como nombres de tablas y columnas, no se pueden escapar, por lo que los nombres de estructura proporcionados por el usuario son peligrosos. Este es un problema común en el software de generación de informes.
- Utilice LIMIT y otros controles SQL dentro de las consultas para evitar la divulgación masiva de registros en caso de inyección SQL. (CHEAT, s.f.)

5.6 Características y Tipos de Inyección SQL:

La inyección SQL ocurre cuando los parámetros proporcionados por el usuario (como los datos en formularios o URLs) no son debidamente validados, lo que permite que comandos maliciosos se incluyan en las consultas SQL. Existen varios tipos de inyección SQL, entre los más comunes se incluyen:

- **Inyección SQL Clásica:** El atacante manipula directamente las entradas de la aplicación para modificar las consultas SQL. Ejemplo: modificar el valor de un campo de entrada para ejecutar un comando SELECT arbitrario.
- **Inyección de Unión (Union-Based SQL Injection):** Permite combinar múltiples consultas SQL en una sola, obteniendo datos de tablas no relacionadas.
- **Inyección Ciega (Blind SQL Injection):** En este tipo de ataque, el atacante no obtiene directamente los resultados de la consulta, pero puede inferir información al observar el comportamiento de la aplicación (por ejemplo, cambios en la respuesta HTTP).

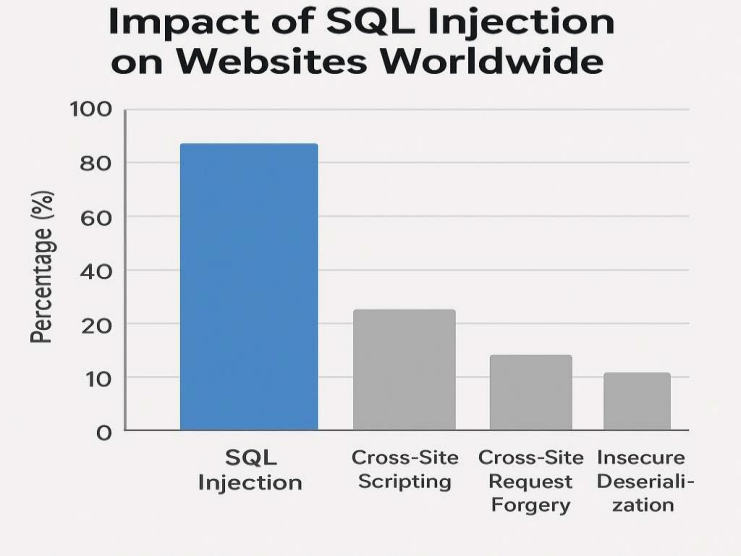
- **Inyección de Tiempo (Time-Based Blind SQL Injection):** El atacante observa la diferencia de tiempos en la respuesta de la aplicación para inferir si la consulta SQL está funcionando como se espera.

El impacto de este tipo de vulnerabilidad es grave, ya que permite al atacante obtener acceso a la base de datos, robar, modificar o borrar información, comprometer la integridad del sistema.

5.7 Impacto de la Inyección SQL en las Organizaciones

El ataque de inyección SQL puede comprometer la disponibilidad, integridad y confidencialidad de los datos. Según Najar Pacheco y Suárez (2015) (scholar.google, 2014), la información es un activo invaluable para las organizaciones, y cualquier ataque que ponga en riesgo la confidencialidad o integridad de esta información puede resultar en daños irreparables. Las consecuencias de una inyección SQL exitosa incluyen:

- **Robo de Datos:** Los atacantes pueden acceder a bases de datos que contienen información sensible, como datos personales, financieros o corporativos.
- **Modificación de Datos:** La alteración de datos puede causar un daño considerable en la operación de una empresa, afectando la toma de decisiones o las relaciones con clientes y proveedores.
- **Eliminación de Datos:** La pérdida de datos críticos puede tener efectos devastadores sobre la operación del negocio y puede resultar en la pérdida de confianza de los clientes.
- **Acceso no Autorizado:** Los atacantes pueden escalar privilegios y obtener acceso completo al sistema, lo que les permite ejecutar comandos maliciosos mar control del servidor.



Impacto de inyección SQL en páginas web (ITD, 2025)

Ilustración 3

La inyección descendió al tercer puesto. El 94 % de las aplicaciones se sometieron a pruebas para detectar algún tipo de inyección, con una tasa de incidencia máxima del 19 %, una tasa de incidencia promedio del 3 % y 274 000 incidencias. Entre las Enumeraciones de Debilidades Comunes (CWE) más destacadas se encuentran CWE-79: Cross-site Scripting, CWE-89: SQL Injection y CWE-73: External Control of File Name or Path.

La prevención de la inyección SQL se basa en buenas prácticas de desarrollo, como el uso adecuado de preparación de consultas (queries preparadas) y procedimientos almacenados, la validación y saneamiento adecuado de las entradas de usuario y la implementación de políticas de control de acceso.

Sin embargo, debido a la complejidad y la constante evolución de los ataques, es fundamental contar con herramientas especializadas que ayuden en la detección temprana y mitigación de las vulnerabilidades. SQLMAPS, que es un marco de pruebas de penetración, se presenta como una herramienta eficaz en este sentido.

5.8 SQLMAPS

Herramienta para la Detección de Inyección SQL de código abierto utilizada para automatizar la detección y explotación de vulnerabilidades de inyección SQL en aplicaciones web.

Se utiliza principalmente para:

- Detectar automáticamente inyecciones SQL.
- Explotar esas vulnerabilidades para extraer información de bases de datos (como usuarios, contraseñas, tablas, etc.).
- Tomar control total del sistema en algunos casos (por ejemplo, obtener acceso al sistema operativo si la base de datos lo permite).

Comparativa con otras herramientas de ciberseguridad

En el análisis de vulnerabilidades web, SQLMap y Acunetix representan herramientas complementarias, pero con enfoques distintos. SQLMap es una herramienta gratuita y open-source especializada en la detección y explotación de inyecciones SQL, ofreciendo un control técnico profundo sobre payloads, bases de datos y niveles de riesgo, pero limitada únicamente a vulnerabilidades de SQL y con una curva de aprendizaje alta. Por su parte, Acunetix es un escáner comercial de aplicaciones web que permite identificar más de 7000 vulnerabilidades, incluyendo XSS, CSRF y errores de configuración, con facilidad de uso, reportes profesionales y capacidad de integración empresarial, aunque no permite explotación directa y su costo es elevado. La elección entre ambas depende del objetivo.

Comparativa entre Acunetix y SQLMap

Herramienta	Tipo	Propósito
SQLMap	Especializada	Explotación y detección profunda de SQL Injection. Muy técnica.
Acunetix	Escáner comercial integral	Detecta muchas vulnerabilidades: XSS, CSRF, RFI, LFI, SQLi, etc. Interfaz profesional, reportes completos.
Tu aplicación	Plataforma educativa/automatizadora	Facilitar auditorías usando SQLMap como backend, enseñando, simplificando y guiando al usuario.

6.1.7 Principales Funciones de SQLMAP:

- Soporta múltiples tipos de bases de datos: MySQL, PostgreSQL, Oracle, Microsoft SQL Server, SQLite, entre otras.
- Puede hacer enumeración de bases de datos, tablas, columnas y datos.
- Puede bypassear WAFs (firewalls de aplicaciones web).
- Soporta distintos tipos de inyección SQL: in-band, blind, out-of-band.

6.1.8 Parámetros de Inyección en SQL y su Prueba con SQLMap

SQLMap es una herramienta automatizada utilizada para detectar y explotar vulnerabilidades de inyección SQL en aplicaciones web. Los ataques de inyección SQL se realizan manipulando los parámetros de la URL o de los formularios de entrada (GET,

POST, etc.), lo que permite a los atacantes modificar las consultas SQL que una aplicación web realiza a su base de datos.

6.1.9 Tipos Comunes de Inyección SQL

Comentarios SQL (--, #, / ... */): Los comentarios SQL permiten interrumpir o alterar la ejecución de la consulta SQL, generalmente para evitar que las secciones no deseadas de la consulta se ejecuten o para omitir las restricciones.

http://example.com/product.php?id=1 --

Operadores SQL (AND, OR, SLEEP (), EXISTS, LIKE, etc.): SQLMap también utiliza operadores para modificar el comportamiento de la consulta SQL. Estos operadores pueden forzar la ejecución de ciertas condiciones lógicas (como AND o OR), o incluso realizar acciones como hacer que la consulta "duerma" por un tiempo, lo que puede ayudar a identificar vulnerabilidades.

http://example.com/product.php?id=1 OR 1=1

6.1.10 Proceso de Inyección SQL con SQLMap

5.8.1.1 Identificación de Parámetros Vulnerables:

Para realizar un ataque de inyección SQL, el primer paso es identificar los parámetros de la URL o los formularios de entrada como: http://example.com/product.php?**id=5**

El parametro Id=5 se pasa a la aplicación web, y puede serutilizado para realizar una consulta a la base de datos como:SELECT * FROM products WHERE id = 5;

5.8.1.2 Manipulación del Parámetro:

Si la aplicación no valida correctamente el valor del parámetro id, un atacante podría manipularlo

inyectando código SQL malicioso. Por ejemplo, cambiando el parámetro id de la siguiente forma:

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

Esto forzaría a la consulta de SQL original en algo como:

SELECT * FROM products WHERE id = 5' OR 1=1 --;

En este caso, la consulta siempre devolverá resultados, ya que 1=1 es siempre verdadero. Esto permite que el atacante obtenga acceso no autorizado a los datos de la base de datos o incluso ejecute comandos maliciosos

¿Por qué es necesario agregar parámetros en la URL?

En muchas aplicaciones web modernas, los parámetros de la URL o de los formularios de entrada son utilizados para generar contenido dinámico, como mostrar productos, filtrar información o consultar registros de la base de datos. Estos parámetros interactúan directamente con las consultas SQL y, si no se validan adecuadamente, pueden ser aprovechados para ejecutar inyecciones SQL maliciosas.

La inyección SQL puede ocurrir si los parámetros no están correctamente validados o sanitizados, lo que permite que un atacante los modifique y ejecute comandos SQL maliciosos en la base de datos, obteniendo acceso a información sensible o manipulando los datos. (GitHub, s.f.) (OWASP, 2025)

5.8.2.1 Explotación de Vulnerabilidad en URL

A. Suponiendo que tienes la siguiente URL

http://example.com/product.php?id=5

B. Si la aplicación no valida adecuadamente el parámetro Id, un atacante podría modificar para inyectar código SQL malicioso de la siguiente manera.

http://example.com/product.php?id=5' OR 1=1 –

C. Esto se comvertiria en una consulta SQL.

SELECT * FROM products WHERE id = 5' OR 1=1 --;

El OR 1=1 siempre es verdadero, lo que hace que la consulta devuelva todos los productos, incluso si el atacante no tiene permisos para verlos

5.8.2.2 Resultado:

[INFO] testing for SQL injection on parameter 'id'

[INFO] parameter 'id' appears to be vulnerable to SQL injection

[INFO] the SQL query that is executed is:

SELECT * FROM products WHERE id = 5' OR 1=1 --;

Explicación del hallazgo:

- **Vulnerabilidad Detectada:** SQLMap confirma que el parámetro id es vulnerable a inyección SQL.
- **Tipo de Inyección:** SQLMap muestra que se puede inyectar código como OR 1=1 -- para manipular la consulta.
- **Consecuencia:** El atacante puede obtener datos no autorizados de la base de datos

Recomendación:

- **Validación y Saneamiento de Entradas**: Asegúrate de validar todos los parámetros de entrada.
- **Uso de Sentencias Preparadas**: Implementa consultas parametrizadas para evitar la manipulación directa de SQL.

5.8.2.3 Validación de parámetros en entradas:

La validación de parámetros es una de las mejores prácticas para evitar vulnerabilidades de inyección SQL y proteger aplicaciones web.

Cada parámetro debe cumplir con el tipo de dato esperado. Si un parámetro debe ser un número entero (por ejemplo, un ID de producto), es importante asegurar que solo contenga números. Si un parámetro id debe ser un número, verifica que no contenga caracteres especiales como comillas, símbolos o letras:

```python
def validate_id(param):
    if param.isdigit():
        return True
    return False
```

PARAMETRO id

Ilustración 4

def validate_id(param): Se define la función Validate_id, que recibe un único parámetro llamado (param), sería la entrada que la función validará.

if param.isdigit(): Este método de cadena de text(string) verifica si todos los caracteres de (param) son dígitos, lo que significa que solo podrá contener números

- Si param es una cadena de texto que solo contiene números, param.isdigit() devolverá True.
- Si param contiene cualquier otro carácter que no sea un dígito (como letras o símbolos), devolverá False

Tipo de validación	Descripción breve	¿Incluido en --risk=1?
Boolean-based (content/ boolean blind)	Inyecta condiciones que camblan lá respuesta (tr-ue/false) para detectar vulnerárábilidad.	AND 1=1 --
Error-based	Provoca errores SQL controlados y busca diferencias en la respe-sas.	') AND (SELECT 1 FROM (SELECT COUNT(*), CONCAT (0x7e, (SELECT user,), 0x7e, FLOOR(RAND(0)*2) x FROM information_schema.tables (
UNION query (básicas)	Inserta consultas UNION SELECT sencillas para extraer columnas	UNION SELECT 1, @@version --
Genéricas no destructivas		-- payloads seguros
Alcance por defecto (parámetros)	Pruebas sobre parámetros comunes (GET/POST) con	-
Time-based (inyección por tiempo)	Usa retardos para detec-tar inyección (más intru-	SLEEP(5) OR 'a'='a ' / agresivo '/

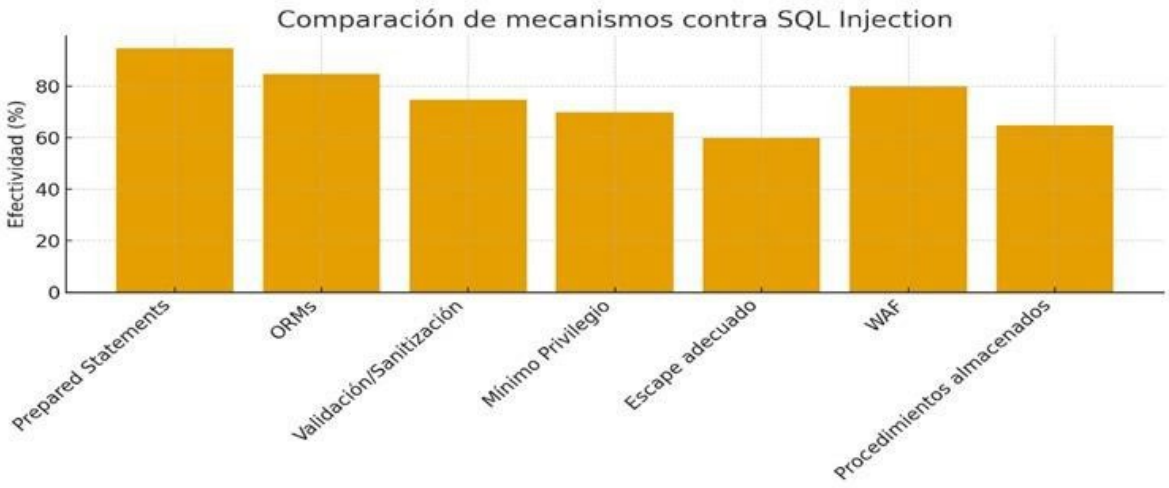
Algunos tipos de validaciones

Ilustración 5

Nivel	Tipo de pruebas	Riesgo	¿Para qué sirve?
1	Básicas, seguras	Bajo	Reconocimiento rápido
2	Moderadas, basadas en tiempo	Medio	Auditorías internas
3	Pesadas, profundas	Alto	Entornos de laboratorio
4	Muy pesadas, evasión avanzada	Muy alto	Stress testing controlado
5	Extremas y experimentales	Máximo	Pruebas de límite en entornos aislados

Niveles de riesgos evaluados por SQLMap

Ilustración 6



Mecanismos de prevencion contra ataques de inyeccion SQL

Ilustración 7

5.9 Marco legal

El desarrollo de un sistema de detección y mitigación de inyección SQL se sustenta en un marco legal compuesto por normativas nacionales (Colombia) e internacionales que establecen obligaciones específicas relacionadas con la protección de datos, la seguridad en el tratamiento de la información y la prevención de incidentes informáticos. Estas disposiciones jurídicas buscan garantizar la integridad, confidencialidad y disponibilidad de los datos, así como promover prácticas responsables en el diseño, implementación y operación de sistemas informáticos. En este contexto, el cumplimiento de dichas normas resulta indispensable, no solo para reducir la exposición a ataques como la inyección SQL, sino también para evitar sanciones administrativas, económicas y penales derivados de la gestión inadecuada de la información y la falta de controles de seguridad. Este marco legal proporciona, por tanto, los lineamientos esenciales que orientan el desarrollo seguro del

proyecto y aseguran su alineación con estándares regulatorios vigentes y buenas prácticas internacionales.

6.1.1 Ley 1581 de 2012 – Protección de Datos Personales (Colombia)

Regula el tratamiento de datos personales en el país y exige implementar medidas técnicas y administrativas para evitar accesos no autorizados, alteración o pérdida de información.

Una inyección SQL constituye un riesgo directo al principio de **seguridad** establecido en la ley.

Consecuencias por incumplimiento:

- Multas hasta por 2.000 salarios mínimos legales vigentes (SMLV).
- Suspensión temporal de operaciones de tratamiento de datos.
- Cierre inmediato del sistema que genere el riesgo.
- Referencia: Congreso de Colombia. (2012). *Ley 1581 de 2012*. Diario Oficial 48.587.

6.1.2 Decreto 1377 de 2013 – Reglamentación de la Ley 1581 (Colombia)

Obliga a implementar controles de seguridad robustos para evitar el acceso indebido a bases de datos.

La ausencia de validación de entradas o de protección ante inyección SQL puede interpretarse como una **falla en la adopción del nivel de seguridad adecuado**.

Consecuencias por incumplimiento:

- Sanciones administrativas por parte de la SIC (Superintendencia de Industria y Comercio).
- Órdenes de corrección inmediata de vulnerabilidades.

Ministerio de Comercio, Industria y Turismo. (2013). *Decreto 1377 de 2013*.

6.1.3 Ley 1266 de 2008 – Habeas Data Financiero (Colombia)

Aplica especialmente a entidades que manejan datos financieros o crediticios. Establece que los administradores de bases de datos deben proteger la integridad y confidencialidad de la información. Una inyección SQL que exponga datos financieros es considerada violación grave.

Consecuencias por incumplimiento:

- Multas económicas de la SIC.
- Cierre temporal del archivo financiero.
- Procesos penales si se demuestra negligencia.

Congreso de Colombia. (2008). *Ley 1266 de 2008*. Diario Oficial 47.219.

6.1.4 Ley 1273 de 2009 – Delitos Informáticos (Colombia)

Tipifica como delito:

- Acceso abusivo a un sistema informático.
- Interceptación de datos.
- Violación de datos personales.
- Daño informático.

Muchos ataques SQL buscan realizar exactamente estas acciones, por lo que fortalecer la protección es parte del cumplimiento legal.

Consecuencias por incumplimiento (para atacantes y administradores negligentes):

- Penas de prisión entre 48 y 96 meses.
- Multas entre 100 y 1.500 salarios mínimos.

Congreso de Colombia. (2009). Ley 1273 de 2009.

6.1.5 Ley 1341 de 2009 – Tecnologías de la Información y las Comunicaciones

Promueve el desarrollo de sistemas TIC seguros y confiables.

Exige que las entidades que implementan sistemas tecnológicos adopten prácticas que minimicen riesgos y garanticen continuidad operativa.

Consecuencias por incumplimiento:

Investigaciones administrativas.

- Sanciones por inadecuada gestión del riesgo tecnológico.
- Congreso de Colombia. (2009). *Ley 1341 de 2009*.

6.1.6 ISO/IEC 27001:2022 – Gestión de Seguridad de la Información

Norma global que exige identificar vulnerabilidades, establecer controles de seguridad y evaluar riesgos. Una aplicación vulnerable a SQLi incumple los controles A.5 (políticas), A.8 (gestión de activos) y A.14 (seguridad en sistemas de información).

Consecuencias por incumplimiento:

- Pérdida o revocación de la certificación ISO.
- Sanciones contractuales con clientes que exigen cumplimiento de estándares.
- Aumento del riesgo reputacional.

International Organization for Standardization. (2022). *ISO/IEC 27001:2022*.

6.1.7 OWASP Top 10 (Internacional)

Guía técnica de referencia obligada en auditorías y desarrollos.

La inyección SQL aparece como una de las vulnerabilidades más críticas por impacto y frecuencia. Las organizaciones que no mitiguen riesgos catalogados por OWASP pueden ser consideradas negligentes en auditorías de seguridad.

Consecuencias por incumplimiento:

- Incumplimiento de buenas prácticas internacionales.
- Riesgo de sanciones contractuales o legales por negligencia técnica.

OWASP Foundation. (2021). OWASP Top 10 – Web Application Security Risks.

5.10 Marco conceptual

6.1.8 Lenguajes, Consultas y Bases de Datos

SQL – Structured Query Language

Lenguaje de consultas utilizado para gestionar bases de datos relacionales.

PL/SQL – Procedural Language / SQL

Extensión procedural de Oracle para lógica dentro de bases de datos.

T-SQL – Transact-SQL

Extensión de Microsoft SQL Server con funciones adicionales al SQL estándar.

ISQL – Inyección SQL

Abreviatura usada para referirse al ataque SQL Injection.

Vulnerabilidades y Seguridad Web

OWASP – Open Web Application Security Project

Fundación internacional reconocida por el OWASP Top 10.

Inyección SQL (SQL Injection)

Ataque que manipula consultas SQL insertando código malicioso.

XSS – Cross-Site Scripting

Vulnerabilidad que permite inyectar scripts maliciosos en páginas web.

CSE – Cross-Site Scripting

Sigla usada en el texto como referencia a XSS.

CWE – Common Weakness Enumeration

Listado de debilidades de software gestionado por MITRE.

CVE – Common Vulnerabilities and Exposures

Identificadores públicos de vulnerabilidades conocidas.

WAF – Web Application Firewall

Firewall especializado en proteger aplicaciones web mediante análisis del tráfico HTTP.

6.1.9 Entidades y Organismos de Ciberseguridad

CSIRT-GOB (Colombia)

Equipo gubernamental encargado de responder a incidentes de ciberseguridad.

CISA – Cybersecurity and Infrastructure Security Agency (EE. UU.)

Agencia responsable de la protección de infraestructura crítica.

FBI – Federal Bureau of Investigation (Cyber Division)

Entidad de EE. UU. que investiga delitos informáticos.

NCSC – National Cyber Security Centre (Reino Unido)

Centro encargado de la protección de sistemas gubernamentales y asesoría en ciberseguridad.

NCA – National Crime Agency (Reino Unido)

Investiga delitos graves, incluyendo cibercrimen.

INCIBE – Instituto Nacional de Ciberseguridad (España)

Institución dedicada a protección, investigación y formación en ciberseguridad.

CCN-CERT – Centro Criptológico Nacional (España)

Equipo estatal especializado en gestión de incidentes en sistemas públicos.

ICO – Information Commissioner's Office (Reino Unido)

Entidad encargada de supervisar protección de datos y privacidad.

ITD – Institute of Technology Development

Institución referenciada como fuente en tu marco teórico.

6.1.10 Tecnologías y Desarrollo del Sistema

API – Application Programming Interface

Interfaz que permite la comunicación entre aplicaciones.

ORM – Object-Relational Mapping

Técnica que permite interactuar con bases de datos mediante objetos.

Backend

Lógica interna del sistema, control de procesos, APIs y comunicación con herramientas externas.

Frontend

Interfaz visual que permite interacción del usuario con la aplicación.

SQLMap

Herramienta automatizada para detectar y explotar vulnerabilidades de SQL Injection.

Arquitectura del Sistema

Diseño que define módulos, flujo de datos e interacción entre componentes.

Flujo de Datos

Ruta que siguen los datos entre backend, frontend y procesos internos.

React

Biblioteca usada para crear interfaces dinámicas.

Gestión de Estados

Control de datos, comportamientos y actualizaciones en interfaces React.

Python

Lenguaje de programación usado para el backend.

Flask

Microframework de Python para construir APIs y manejar rutas.

Endpoints REST

Rutas del backend accesibles mediante solicitudes HTTP.

Procesos en Background

Tareas que se ejecutan sin bloquear la interacción del usuario.

JSON

Formato utilizado para almacenar y transferir datos entre módulos del sistema.

Reportes HTML

Documentos generados automáticamente para presentar resultados de manera visual.

6.1.11 HTTP y Parámetros del Escaneo

GET – Método GET

Solicitudes HTTP que envían parámetros en la URL.

POST – Método POST

Solicitudes HTTP que envían datos en el cuerpo del mensaje.

URL – Uniform Resource Locator

Dirección usada para acceder a recursos web.

Método HTTP

Tipo de solicitud utilizada durante la configuración del escaneo.

Nivel de Riesgo

Determina qué tan agresivo será el escaneo.

Nivel de Análisis

Profundidad y variedad de pruebas aplicadas por SQLMap.

6.1.12 Entornos y Pruebas

Página Web Vulnerable

Sitio diseñado intencionalmente con fallas de seguridad para ejecutar pruebas.

Entorno Local

Espacio de pruebas aislado del internet y controlado.

Pruebas Unitarias

Validación de funciones específicas del backend.

API – Frontend

Comunicación entre la interfaz y el backend durante el análisis.

5.10.6 Metodologías del Proyecto

SCRUM

Metodología ágil usada durante el desarrollo.

Transparencia

Claridad en tareas, roles y avances.

Inspección

Revisión periódica del progreso del proyecto.

Adaptación

Ajustes necesarios para mejorar procesos o corregir errores.

Bitrix24 / Jira

Plataformas usadas para organizar tareas y registrar avances.

6.1.13 Objetivos Globales y Sociales

ODS – Objetivo de Desarrollo Sostenible

Metas globales de la ONU en educación, investigación, seguridad y desarrollo tecnológico.

6. Metodología

6.1 Tipo de investigación

El presente proyecto se enmarca en la Investigación Aplicada debido a su orientación hacia el desarrollo de una solución tecnológica funcional destinada a la detección y evaluación automatizada de vulnerabilidades de SQL Injection en páginas web, utilizando la herramienta SQLMap. Se adopta por un Enfoque Cuantitativo para la recopilación de datos objetivos y medibles (como niveles de severidad), complementado con una Metodología Experimental que permitirá comprobar la eficacia del sistema propuesto a través de pruebas rigurosas en un entorno controlado y simulado.

6.2 Área de estudio

El proyecto se enmarca en el área de Informática y Seguridad de la Información, específicamente dentro del campo:

- Análisis de vulnerabilidades
- Pruebas de penetración (Pentesting)
- Automatización de seguridad en aplicaciones web
- Desarrollo de software backend y frontend

6.3 Desarrollo Metodológico

Para el desarrollo de este proyecto se llevaron a cabo los siguientes pasos

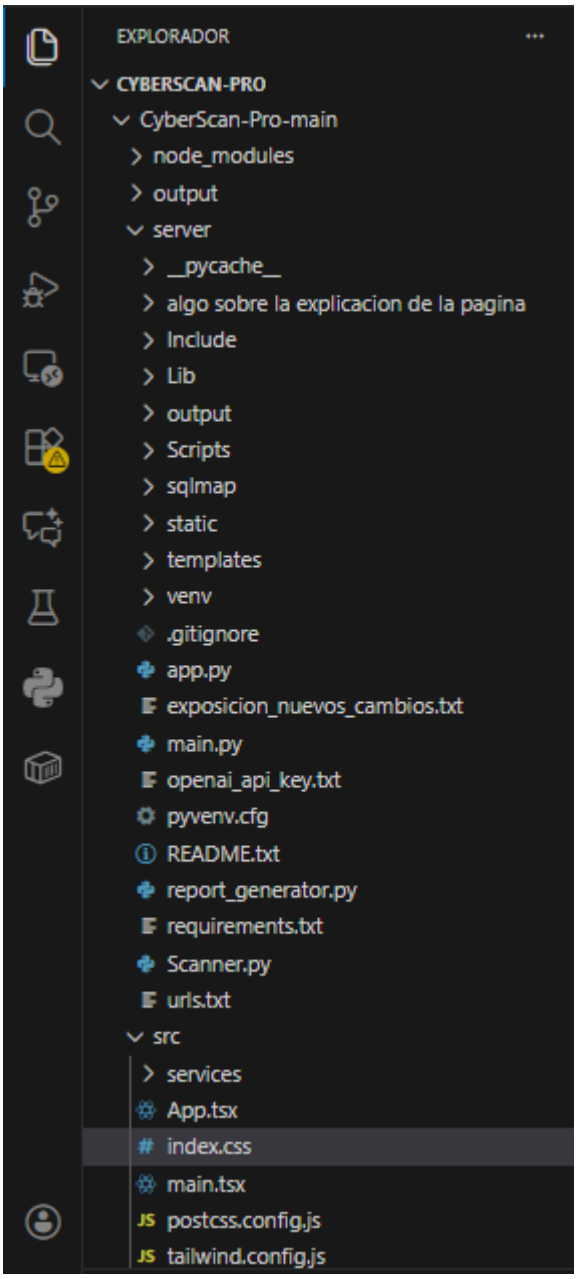
Paso 1. Análisis y Revisión Técnica

Se realizó una revisión técnica de las herramientas y metodologías existentes para la detección de vulnerabilidades web, tales como SQLMap y técnicas OWASP. Esto permitió

identificar los componentes fundamentales para construir el backend, los parámetros necesarios para ejecutar escaneos, y los procesos internos que permiten evaluar el nivel de riesgo.

Paso 2. Diseño del sistema

Con la información técnica recopilada, se procedió a diseñar la arquitectura general del sistema, incluyendo el flujo de datos entre los módulos, la comunicación con SQLMap, la gestión de estados en React y la estructura del reporte generado. Este diseño incluyó la definición del comportamiento del usuario, las pantallas necesarias y el modelo de interacción entre componentes.

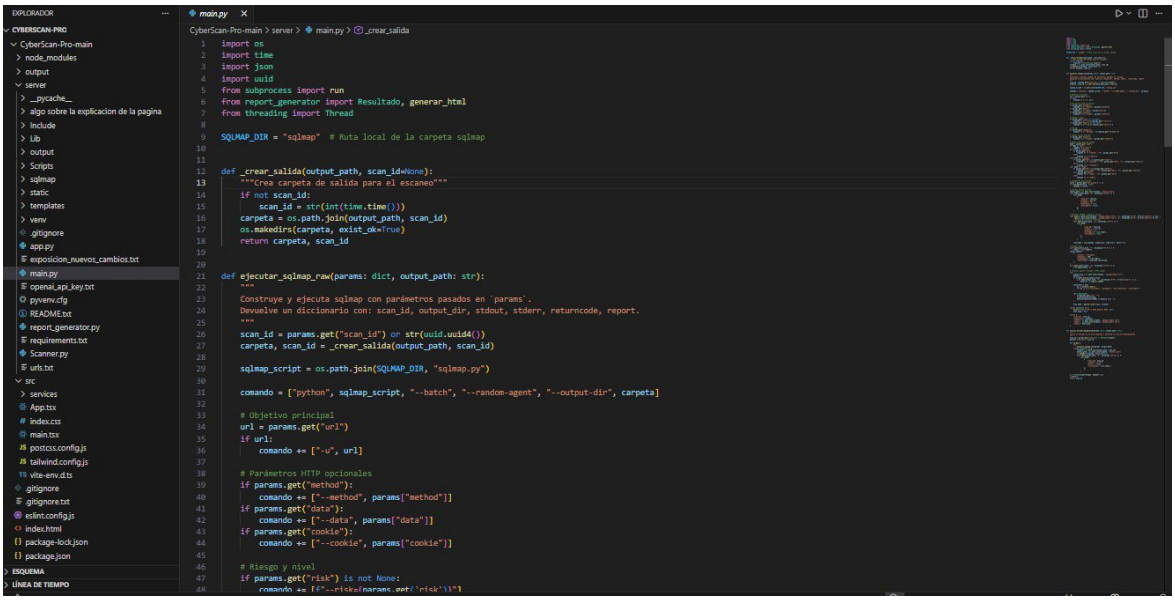


Ilustracion 8 Python

Paso 3. Desarrollo del Backend

El backend se desarrolló utilizando Python y el framework Flask. Este módulo se encarga de gestionar las solicitudes de escaneo, enviar los parámetros a SQLMap, manejar procesos en background y generar reportes del análisis. Entre las tareas implementadas se incluyen:

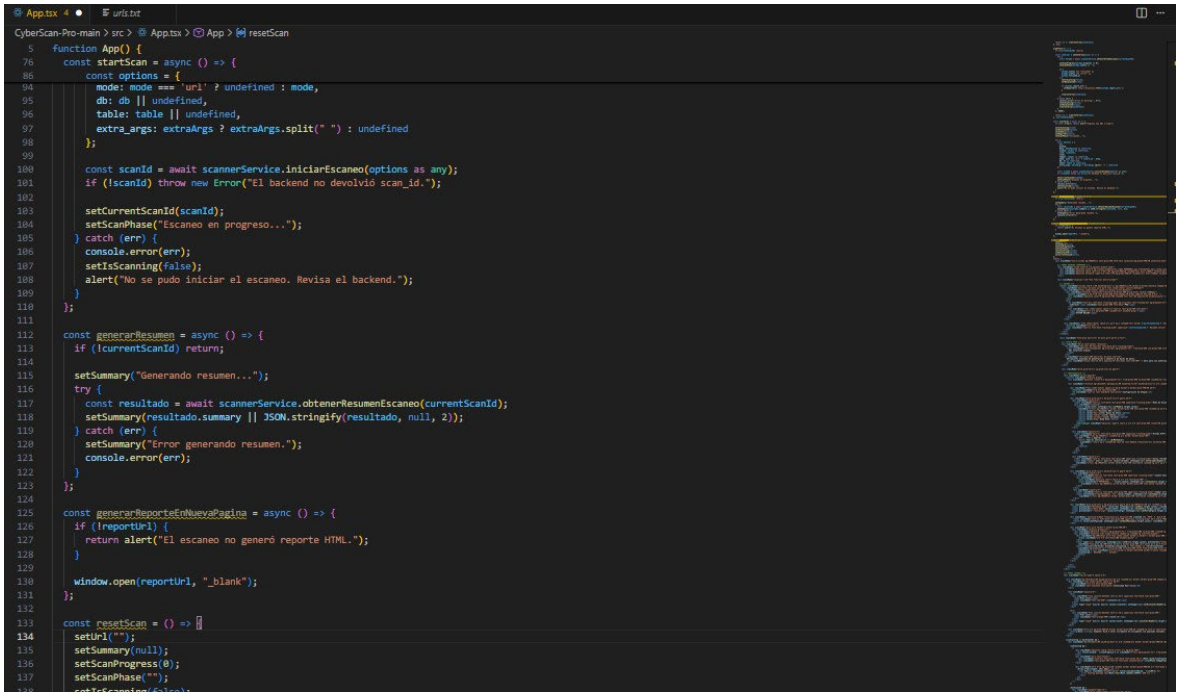
- Creación de endpoints REST.
- Ejecución de SQLMap mediante comandos automatizados.
- Manejo de archivos JSON de resultados.
- Generación de reportes legibles.
- Control del estado del escaneo y envío de respuestas al frontend.



Ilustracion 9 codigo Python

Paso 4. Desarrollo del Frontend

El frontend se desarrolló empleando React. Se implementaron pantallas dinámicas para la configuración del escaneo, visualización del progreso, manejo de errores y lectura del reporte final. Se integraron bibliotecas de iconos (Lucide-React), hooks personalizados y animaciones básicas. El frontend permite al usuario configurar parámetros como método HTTP, nivel de riesgo, nivel de análisis, entre otros.



Ilustracion 10 codigo python

Paso 5. Creación del entorno de pruebas

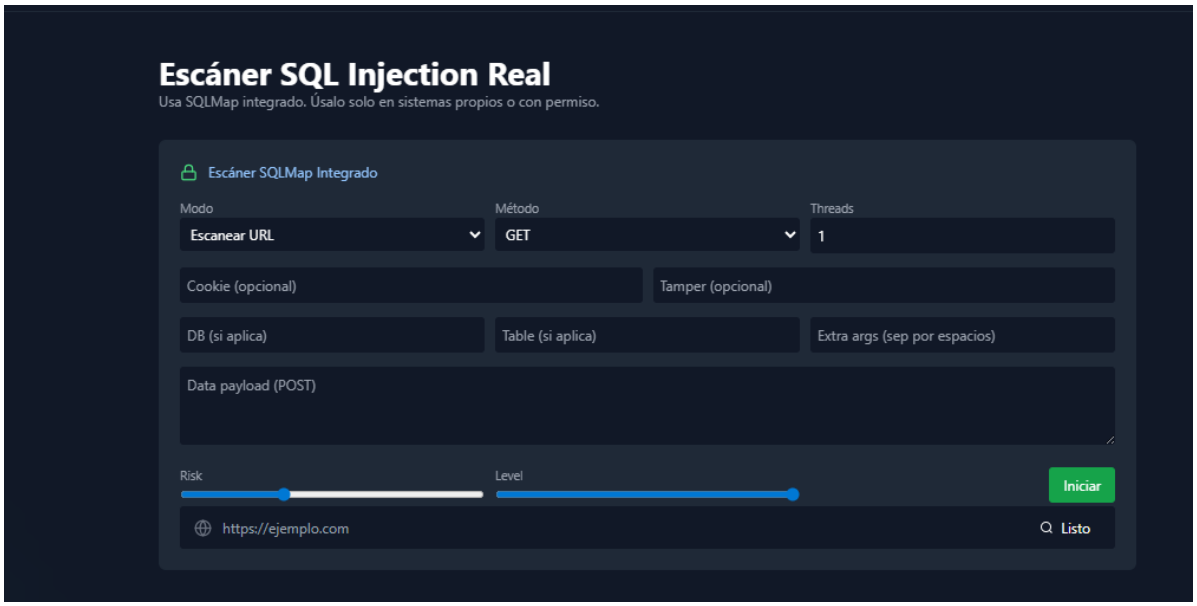
Como primer paso, se desarrollará una página web vulnerable diseñada específicamente para este proyecto. Esta página servirá como entorno de pruebas controlado, donde se simularán ataques de inyección SQL de forma segura. Todo el

proceso se llevará a cabo en un entorno local, evitando comprometer páginas reales, datos sensibles o sistemas en producción.

Paso 6. Pruebas y Depuración del Sistema

Una vez implementadas las funcionalidades principales, se llevaron a cabo pruebas exhaustivas para validar el correcto funcionamiento del escáner. Estas pruebas incluyeron:

- Pruebas unitarias del backend.
- Pruebas del flujo de comunicación API–frontend.
- Simulaciones de escaneo en sitios vulnerables controlados.
- Validación de la interfaz gráfica y del estado del proceso.

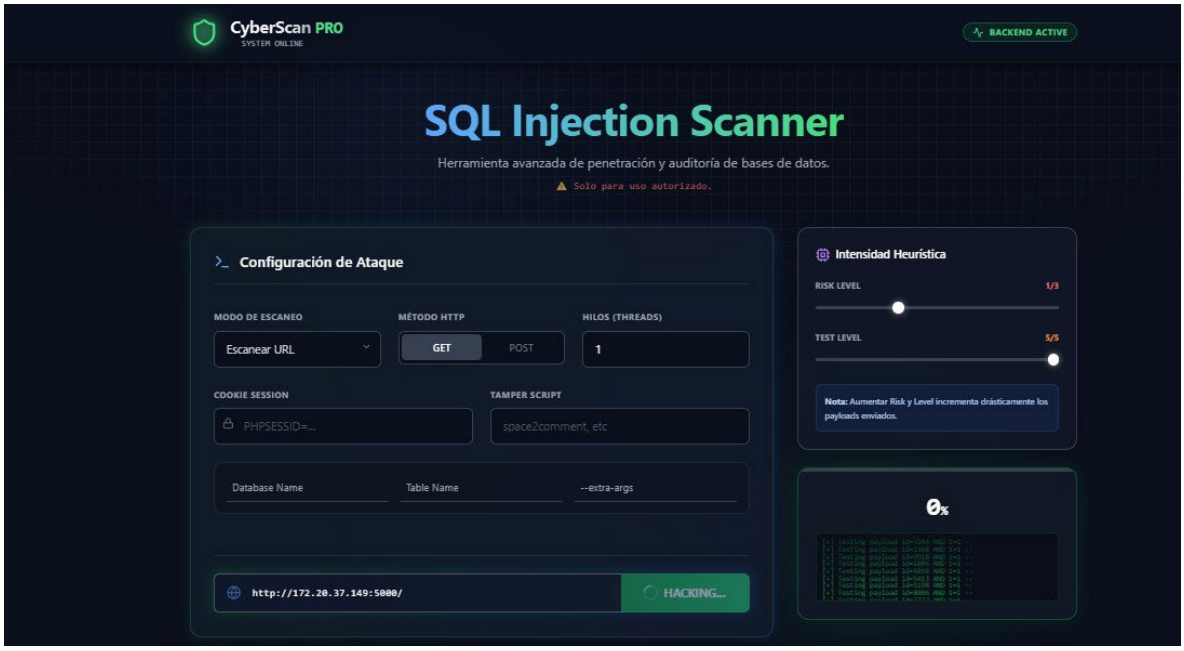


Ilustracion 11 pruebas

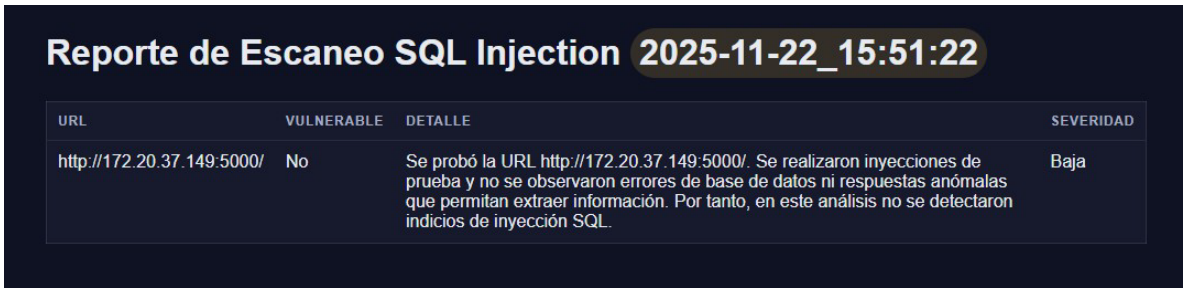
Con base en los errores encontrados (como funciones duplicadas, estados inconsistentes y errores en la captura de resultados), se realizaron correcciones para garantizar la estabilidad del sistema.

Paso 7. Ejecución de ataques controlados

Ya con la herramienta desarrollada, se realizarán simulaciones de ataques sobre la página de prueba. Estos ataques serán de tipo inyección SQL y se aplicarán bajo condiciones seguras, siempre respetando los principios éticos. Lo que se busca es comprobar qué tan bien responde el sistema frente a diferentes escenarios de vulnerabilidad.



Ilustracion 12 con correciones



Ilustracion 13 Resultado

6.1.1 Metodología SCRUM

Los principios scrum utilizados en este proyecto fueron

6.1.1 Trasparencia

Desde el principio, se definieron roles y responsabilidades específicas dentro del equipo, y se utilizó un tablero digital compartido en Bitrix24 y Jira para registrar tareas como: formulación del problema, redacción de objetivos, construcción del marco teórico, elaboración de la metodología y estructuración de la propuesta técnica. Cada tarea incluyó responsable, fecha estimada de entrega y estado de avance, lo cual permitió un seguimiento claro y conjunto.

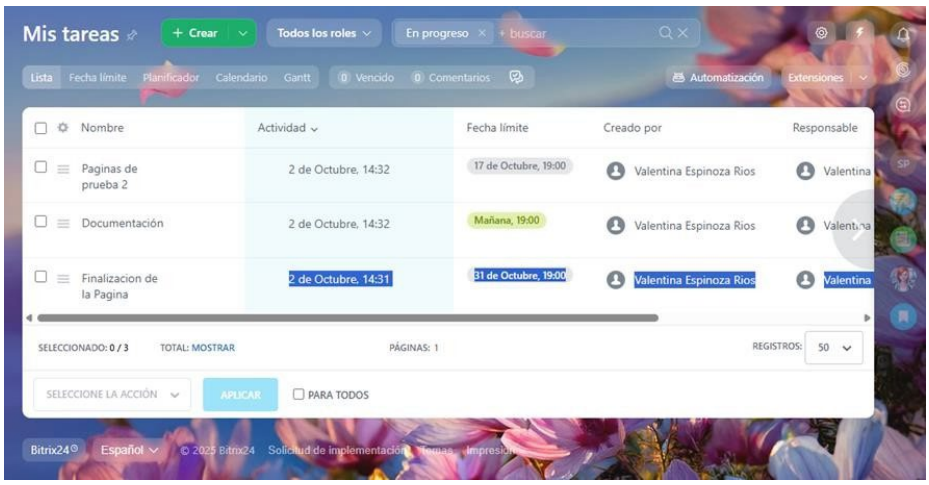
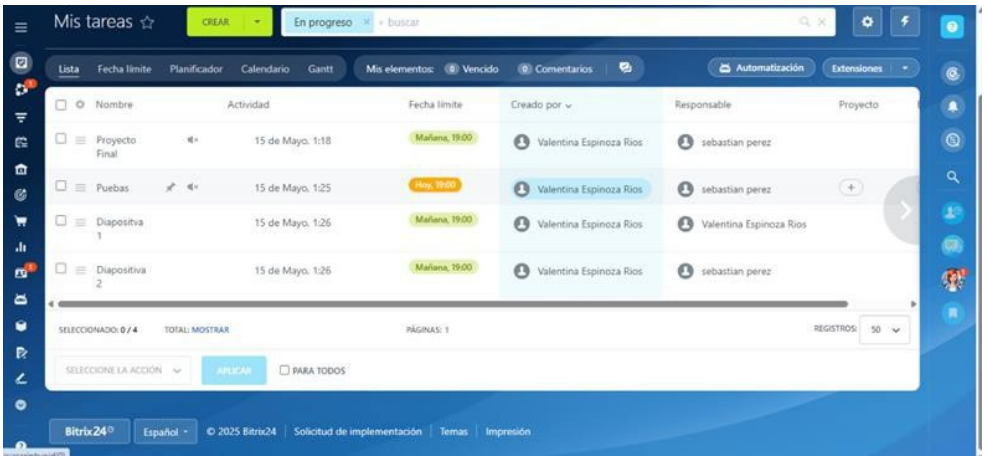
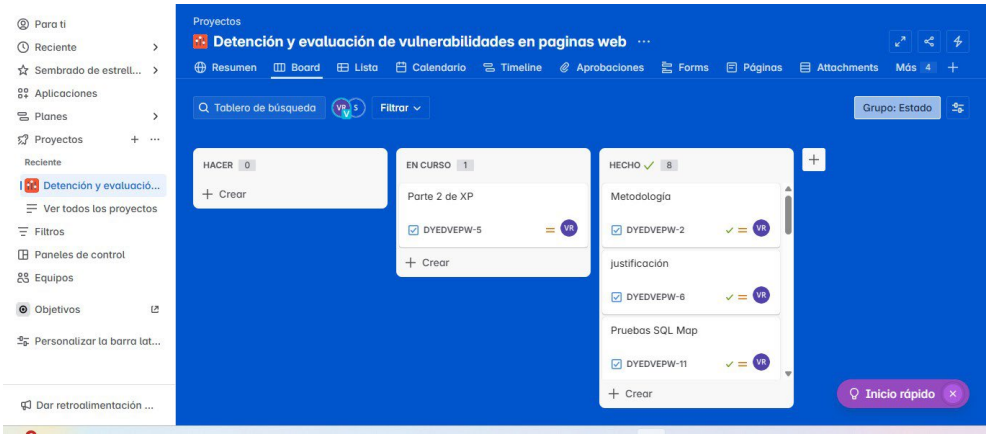


Ilustración 14 Asgnacion de taeras en Britrix24



Ilustracion 15 asignacion de tareas en Britrix24



Ilustracion 16 asignacion de tarea Jira

6.1.2 Inspección

Durante todo el proceso del proyecto, realizamos reuniones para evaluar cómo íbamos. Estas reuniones eran espacios para revisar lo que habíamos logrado, mostrar avances reales (como los primeros reportes generados por el script de automatización) y recibir retroalimentación entre nosotros, igual para dar opiniones sobre lo aportado en el documento.

6.1.3 Adaptación

Gracias a la inspección constante, supimos adaptarnos a los desafíos que fueron surgiendo. Por ejemplo, al principio pensábamos hacer pruebas sobre una página web ya existente, pero por temas legales, decidimos crear páginas simuladas especialmente diseñadas para ser vulneradas de una forma controlada.

- **Elaboración de documentación**

Finalmente, se generará una documentación técnica completa, pensada para facilitar el uso del sistema por parte de otros estudiantes, desarrolladores o personas interesadas en temas de seguridad web. Este documento servirá como evidencia del proceso investigativo y práctico realizado, y como material de consulta para quienes deseen entender cómo detectar y mitigar ataques de inyección SQL utilizando entornos de prueba controlados.

7. Resultados.

Durante las fases de pruebas, se han ejecutó pequeños scripts con Python con el propósito de automatizar el uso de SQLMap. Este script permite lanzar análisis directamente sobre ciertas URLs de prueba, y al finalizar, genera automáticamente un reporte en formato HTML con los resultados obtenidos. En nuestro caso, probamos con nuestra propia página web de prueba.

Gracias a esta automatización, fue posible analizar de forma rápida si existían vulnerabilidades por inyección SQL. Por ejemplo, uno de los análisis mostró que la URL correspondiente a nuestro entorno local (http://localhost:3000/app) era vulnerable, mientras que la otra URL de prueba, https://example.com/product.php?id=1, no presentaban riesgo. Esta información quedó reflejada visualmente en el reporte HTML generado, lo cual facilita la interpretación de los resultados y la posterior documentación

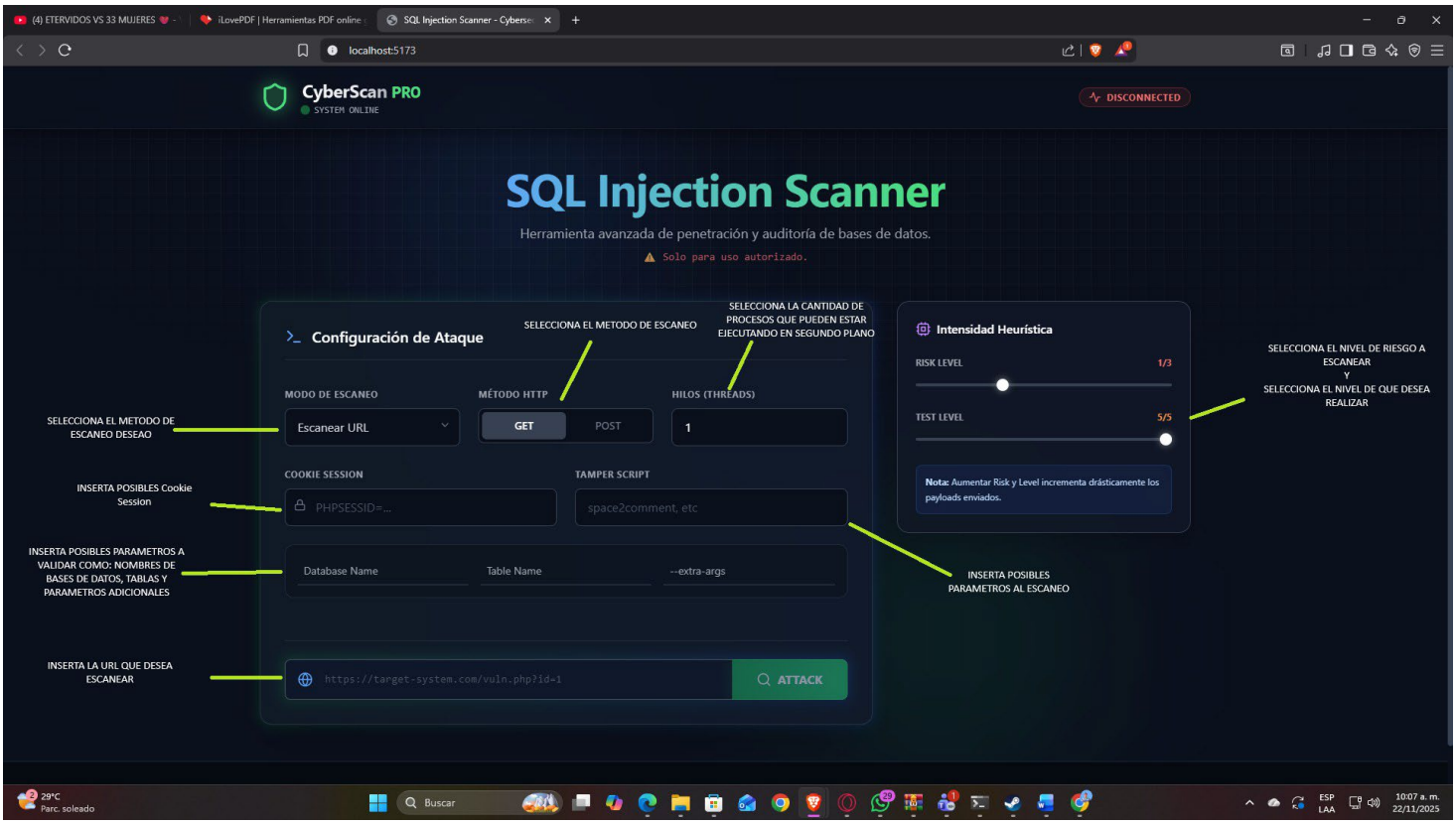
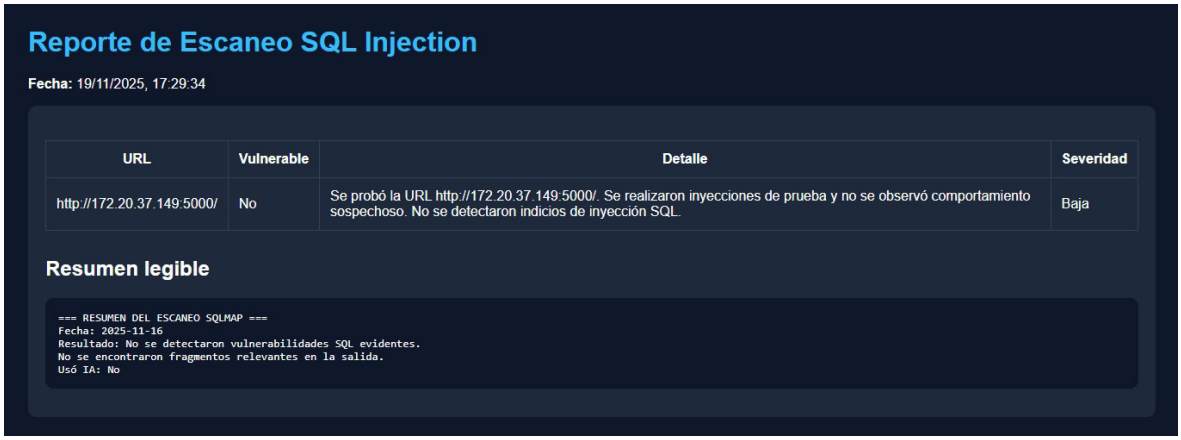


Ilustración 17 de la Página de inicio de SQL Injetion



Ilustracion 18 de los Resultados del Escaneo

7.1 Impacto.

Aunque el proyecto aún está en desarrollo ya se pueden percibir algunos impactos significativos, especialmente en el plano del aprendizaje y la toma de conciencia sobre la seguridad web. Más que solo construir una herramienta, esta experiencia nos ha llevado a entender con mayor claridad la importancia de prevenir ataques como la inyección SQL, que a menudo se subestiman, pero pueden causar daños importantes tanto a nivel personal como organizacional.

Durante las pruebas iniciales realizadas en un entorno simulado que desarrollamos para este fin, hemos podido observar de manera práctica cómo funcionan estos ataques y cómo una herramienta como SQLMap puede ayudar a identificarlos. Esto ha sido clave para conectar los conceptos teóricos con escenarios reales, sin poner en riesgo ningún sistema productivo ni datos sensibles.

7.1 Nivel de madurez

Ilustración 19 Technology Readiness levels (ayming, s.f.)

Nos encontramos en un nivel de madurez de TRL 4 ya que se ha validado la herramienta de análisis (SQLMap), su configuración y la generación automatizada de reportes en un entorno controlado de laboratorio. Además, hemos creado varias páginas web con vulnerabilidades, en la cual se están realizando pruebas de una manera controlada y segura.

7.2 Desarrollo Sostenible Aplicada al Proyecto

Los Objetivos del ODS que están más orientados a este proyecto son:

1.4.4. De aquí a 2030, aumentar considerablemente el número de jóvenes y adultos que tienen las competencias necesarias, en particular técnicas y profesionales, para acceder al empleo, el trabajo decente y el emprendimiento.

2.9.5. Aumentar la investigación científica y mejorar la capacidad tecnológica de los sectores industriales de todos los países, en particular los países en desarrollo, entre otras cosas fomentando la innovación y aumentando considerablemente, de aquí a 2030, el número de personas que trabajan en investigación y desarrollo por millón de habitantes y los gastos de los sectores público y privado en investigación y desarrollo.

3.16.4. De aquí a 2030, reducir significativamente las corrientes financieras y de armas ilícitas, fortalecer la recuperación y devolución de los activos robados y luchar contra todas las formas de delincuencia organizada.

Aporte a la comunidad.

- Prevención de delitos cibernéticos mediante prácticas éticas y educativas, reduciendo la exposición a riesgos digitales en instituciones educativas y organizaciones.

- Fortalecimiento del conocimiento técnico local en seguridad informática, brindando a estudiantes y profesionales herramientas reales para el análisis y protección de sitios web.
- Contribución al desarrollo profesional en áreas estratégicas como ciberseguridad, automatización.
- Impacto positivo en el entorno digital regional, especialmente en zonas donde la protección digital y la formación en seguridad es muy limitada.

Posibilidad de transferir el conocimiento a comunidades vulnerables, instituciones educativas públicas, promoviendo el uso responsable y seguro de tecnologías digitales.

9. Referencias

(s.f.). Obtenido de ayming: https://www.ayming.es/recursos/opinion-experta/trl-technology-readiness-levels/

(Najar Pacheco & Suárez Suárez,). (2015).

Agency, A. C. (s.f.). Obtenido de https://www.cisa.gov/about

Avast. (s.f.). Obtenido de https://www.avast.com/es-es/c-sql-injection#:~:text=y%20c%C3%B3mo%20funciona?,puede%20proteger%20contra%20las%20consecuencias.&text=Este%20art%C3%ADculo%20contiene-,%C2%BFQu%C3%A9%20es%20la%20inyecci%C3%B3n%20de%20SQL?,ataque%20de%20inyecci%C3%B3n%20de%20

CHEAT, O. (s.f.). Obtenido de https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html

Cia. (s.f.). Obtenido de https://www.cia.gov/

Colombia, P. N. (s.f.). Obtenido de https://www.policia.gov.co/directorio/centro-cibernetico-policial

GitHub. (s.f.). Obtenido de https://github.com/sqlmapproject/sqlmap/wiki/Introduction

incibe. (s.f.). Obtenido de https://www.incibe.es/index.php/incibe/informacion-corporativa/que-es-incibe

ITD. (30 de abril de 2025). Obtenido de https://informatecdigital.com/inyeccion-sql-que-es/

OWASP. (2025). Obtenido de https://www.owasptopten.org/

repository.udistrital. (s.f.). Obtenido de https://repository.udistrital.edu.co/items/562cae14-0545-4905-a631-c7f1d036e6c1

scholar.google. (19 de 12 de 2014). Obtenido de https://scholar.google.com/citations?view_op=view_citation&hl=es&user=UXW0J5kAAAAJ&citation_for_view=UXW0J5kAAAAJ:u5HHmVD_uO8C

techradar. (s.f.). Obtenido de https://www.techradar.com/news/national-cyber-security-centre-what-is-it

Tecnología, I. R. (junio de 2021). *scielo*. Obtenido de scielo: http://scielo.senescyt.gob.ec/scielo.php?pid=S1390-860X2021000100104&script=sci_arttext#ref13

tecnology, N. i. (s.f.). Obtenido de https://nvd.nist.gov/general/nvd-dashboard

theguardian. (s.f.). Obtenido de https://www.theguardian.com/business/2016/oct/05/talktalk-hit-with-record-400k-fine-over-cyber-attack

Universidad Central del Sur (CHINA) POBOX 410083, D. d. (11 de julio de 2010). Obtenido de ieeexplore: https://ieeexplore.ieee.org/document/5565202

verizon. (s.f.). Obtenido de https://www.verizon.com/business/resources/reports/dbir/

welivesecurity. (31 de julio de 2012). Obtenido de welivesecurity: https://www.welivesecurity.com/la-es/2012/07/31/vulnerabilidades-estadisticas-e-impacto/

WikipediA. (s.f.). Obtenido de https://en.wikipedia.org/wiki/National_Cyber_Security_Centre_%28United_Kingdom%29

wiz. (s.f.). Obtenido de https://www.wiz.io/vulnerability-database/cve/cve-2025-26590

wnumeration, c. w. (s.f.). Obtenido de https://cwe.mitre.org/data/definitions/20.html