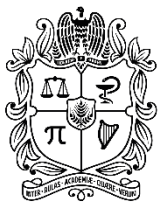


UNIVERSIDAD
NACIONAL
DE COLOMBIA

Diseño de una herramienta automatizada para las pruebas de penetración informática del riesgo de inyecciones SQL inferenciales existente en aplicaciones empresariales bajo ambiente web

Alexis Miguel Taborda Salazar

Universidad Nacional de Colombia

Facultad de Administración, Departamento de Informática y Computación

Manizales, Colombia

2018

Diseño de una herramienta automatizada para las pruebas de penetración informática del riesgo de inyecciones SQL inferenciales existente en aplicaciones empresariales bajo ambiente web

Alexis Miguel Taborda Salazar

Tesis presentada como requisito parcial para optar al título de:

Magister en Administración de Sistemas Informáticos

Director (a):

PhD. Francisco Javier Valencia Duque

Línea de Investigación:

Control y Auditoria de Tecnologías de Información

Grupo de Investigación:

Teoría y Gestión de Tecnologías de Información

Universidad Nacional de Colombia

Facultad de Administración, Departamento de Informática y Computación

Manizales, Colombia

2018

A mi esposa

Su amor ha sido fundamental para darle a mi vida estabilidad. Su paciencia ha sido fundamental para la consecución de mis metas. Su presencia ha sido fundamental para mi existencia.

Agradecimientos

Es indispensable mencionar que agradezco la labor realizada por el PhD. Francisco Javier Valencia Duque, profesor adscrito al departamento de informática y computación de la Facultad de Administración en la Universidad Nacional de Colombia Sede Manizales, quien durante más de dos años usó su conocimiento para dar dirección a mis ideas, revisar mis propuestas y trabajar conmigo en la estructuración y publicación de los artículos generados en este trabajo de grado.

Igualmente, agradezco al profesor Eduardo José Villegas Jaramillo, quien al momento de escribir este documento se desempeñaba como Director Académico de la Universidad Nacional de Colombia Sede Manizales, quien me ánimo a participar en la maestría a la cual esta tesis está asociada.

Y todo lo investigado no hubiera sido posible sin el apoyo del señor vicerrector de la Universidad Nacional de Colombia Sede Manizales, el profesor German Albeiro Castaño Duque, quien aprobó que dedicara tiempo de mis actividades laborales para las actividades de investigación y participación en eventos asociados a este trabajo.

Resumen

De los riesgos de seguridad en ambientes web, el riesgo de inyecciones SQL es catalogado como el más importante, y, de los siete tipos existentes de inyecciones SQL, las inyecciones SQL inferenciales son las que presentan una mayor complejidad en sus pruebas de penetración, debido a que es necesario extraer la información no de manera determinística sino infiriendo los datos al observar cambios de comportamiento en el ambiente web.

La presente tesis de maestría tiene como objetivo la reducción de tiempo empleado en las pruebas de penetración usadas para la evaluación del riesgo de inyecciones SQL inferenciales presente en ambientes web. Para esto se usó una metodología mixta: cuantitativa para el análisis de los algoritmos usados en la evaluación de dicho riesgo y cualitativa al analizar las estrategias y herramientas utilizados actualmente en las pruebas de penetración sobre el riesgo mencionado.

Como resultado de esta investigación, se pudo establecer que el algoritmo bit a bit es el más eficiente en la extracción de información y la herramienta SQLMAP la más completa para su evaluación. En términos de tiempos, la herramienta SQLBrute - SQL Injection Brute Forcer es la mejor para el caso de las inyecciones SQL inferenciales basadas en tiempo y The Mole para el caso de las no basadas en tiempo. Además, se realizó una herramienta usando el algoritmo bit a bit optimizado, y se comparó los tiempos empleados por ella con las herramientas más eficientes disponibles. Al comparar los tiempos de la herramienta desarrollada, se comprobó que esta investigación disminuyo los tiempos empleados en la evaluación del riesgo tratado y que la herramienta desarrollada realiza la extracción de datos de manera más eficiente.

Palabras clave: Inyecciones SQL, Inyecciones Ciegas, Inferencial, Seguridad Web, Riesgos, Pruebas de Penetración, Análisis de vulnerabilidad.

Design of an automated tool for penetration testing on inferential SQL injections risks in business applications under web environment

Abstract

There are many security risks in a web environment. However, SQL injections is the most important risk. This risk has seven sub-types and inferential SQL Injection is the most complex sub-type: It is necessary extract the information not in a deterministic way but inferring the data by means of observing behavior changes in the web environment.

This thesis is a research process on time reduction of inferential SQL injections present in web environments. With that purpose, a mixed methodology was used (quantitative and qualitative) for analyze the strategies, algorithms and tools that penetration testers current use in evaluation of the aforementioned risk. According to the study conducted, bit-to-bit is the faster algorithm for extracting information and SQLMAP tool is the most complete. In time terms, the SQLBrute - SQL Injection Brute Forcer tool is the best tool for based on time inferential SQL injections and The Mole for non-time based injections. After this identification, a tool was made using optimized algorithm bit-to-bit. That tool was compared with the aforementioned tools and verifying that it is effectively more efficient. Finally, the tool was used in a real environment. In this way, it was found that it is possible to decrease the time in the evaluation of the treated risk.

Keywords: SQL Injection, Blind, Inferential, Web Security Risk, Penetration Testing, Vulnerability Analysis

Contenido

Lista de figuras

Pág.

Lista de tablas

Pág.

Lista de Símbolos y abreviaturas

Símbolos con letras latinas

Símbolo	Término
T(n)	Tiempo de ejecución
n	Cantidad de caracteres a extraer
A	Tamaño del Subconjunto de caracteres UNICODE
s	Segundos

Símbolos con letras griegas

Símbolo	Término
α	Cantidad de Caracteres UNICODE
δ	Tiempo de carga de la página
Δ	Tiempo inyectado

Superíndices

Superíndice	Término
i	Exponente. Número de iteraciones.
m	Exponente. Número máximo de iteraciones

Abreviaturas

Abreviatura	Término
$\log_2$	Logaritmo en base dos

Abreviatura	Término
MS	SQL Server
PG	PostgreSQL
MY	MySQL
ORA	ORACLE
Win10	Windows 10
Win7	Windows 7
BS	Búsqueda secuencial
NTB	No basado en tiempo
TB	Basado en tiempo
URL	Uniform Resource Locator
VI	Entrada vulnerable
Nee	Needle, parámetro a buscar cómo cambio en la página web. Sección 4.3.2

Introducción

El autor de ciencia ficción William Gibson acuñó el término ciberespacio como un mundo artificial infinito donde los humanos navegan en un espacio basado en la información (Benedikt, 1992). Al ser Internet el ciberespacio más accedido, la información que allí se encuentra es muy importante.

Esta información es accedida a través de navegadores que consumen ambientes web (Berners-Lee, 1992). Estos ambientes, en su gran mayoría, acceden a bases de datos que contienen la información necesaria operar. Los estándares y la ley colombiana, muestran que es responsabilidad del dueño de dichos entornos garantizar la seguridad de la información que se administre (Ley estatutaria 1581 de 2012). Esto significa mantener su confidencialidad, integridad y disponibilidad (ISO/IEC, 2013).

Algunas organizaciones internacionales realizan análisis de los incidentes de seguridad en los ambientes web y generan rankings sobre las vulnerabilidades críticas en los cuales se debe concentrar un administrador de uno de estos ambientes con el fin de minimizar los riesgos sobre la información. Como ejemplo de esto, se encuentra el proyecto abierto de seguridad en aplicaciones web (OWASP por sus siglas en inglés), una organización pionera en este tema, quien determinó que la vulnerabilidad con mayor criticidad son las inyecciones de código no autorizado (OWASP, 2017).

Actualmente, se cuentan con herramientas automatizadas que identifican y evalúan la vulnerabilidad antes mencionada (Slaviero, 2012). Esta tesis de maestría se enfoca en reducir el tiempo que se emplea en las pruebas de penetración para evaluar las inyecciones SQL inferenciales. Para tal fin, se analiza los algoritmos, técnicas y herramientas usadas y se propone una herramienta que, tanto en la teoría como en la práctica, realiza la evaluación en menor tiempo y de manera exhaustiva. Como resultado, este documento presenta nuevos conceptos sobre el riesgo de inyecciones SQL inferenciales, establece las condiciones matemáticas para el uso de los algoritmos y muestra los pasos empleados en la reducción del tiempo en las pruebas de penetración.

Al realizar esta investigación, se redujo un 13% el tiempo empleado en las pruebas de penetración de las inyecciones inferenciales no basadas en tiempo y 20% sobre las pruebas en inyecciones inferenciales basadas en tiempo y se generó una herramienta que aplica los nuevos conceptos documentados en este trabajo.

El primer capítulo de este documento presenta de manera formal el diseño de este proyecto de maestría, indicando la pregunta de investigación, metodología, objetivos, alcance, justificación y otras generalidades de esta tesis. En el segundo capítulo se encuentra una explicación de las inyecciones SQL inferenciales y un marco teórico sobre el mismo. En el capítulo 3 se encuentra un análisis sobre los algoritmos usados para la identificación del riesgo antes mencionado y el capítulo 4, realiza un análisis sobre las herramientas usadas. Posteriormente, en el capítulo 5, se explica la herramienta propuesta y el análisis y comparativo de la herramienta en un entorno de pruebas, para posteriormente, aplicarlo en un entorno de producción. Finalmente, el capítulo 6 presenta las conclusiones y el análisis de resultados sobre la investigación realizada.

1. Presentación de la Tesis

1.1 Problemática

De acuerdo con (OWASP, 2017), el mayor riesgo de seguridad en los ambientes web son las inyecciones de código no autorizado. Estas inyecciones pueden ser desarrolladas en lenguajes interpretados como código Lightweight Directory Access Protocol (LDAP), ORM (Hibernate, NHibernate, ActiveRecord, EZPDO, etc.), lenguaje de comandos propios del sistema operativo, SQL - Structured Query Language y NOSQL - No Structured Query Language.

Aunque en 1998, (Forristal, 1998) documentó el riesgo de inyecciones SQL, su presencia sigue siendo común en los ambientes web, tal como se puede observar en los informes de OWASP. Este riesgo de seguridad de fácil explotación impacta sobre los tres pilares de la seguridad de la información: Confidencialidad, Integridad y Disponibilidad (OWASP, 2017). A pesar del tiempo y la evolución tecnológica, este riesgo sigue estando presente (Bandhakavi, Bisht, Madhusudan, & Venkatakrishnan, 2007). La comunidad académica ha definido 7 tipos de inyecciones SQL: Tautológico, Consultas Ilegales/Lógicamente Incorrectas, Uniones, Consultas a cuestas, Procedimientos almacenados, Inferenciales y Codificación Alternativa (W. G. J. Halfond, Viegas, & Orso, 2008). De todos los tipos, la evaluación de las inyecciones inferenciales presenta una mayor complejidad debido a que se debe extraer carácter por carácter de la base de datos, lo cual incide en el tiempo de evaluación (Štampar, 2016). En este artículo se indica que para la extracción de 537 bits se puede llegar a tomar 882.84 segundos. Si tomamos en cuenta estos tiempos, la extracción de un solo campo texto que almacene 500 caracteres puede tomar más de 6500 segundos, el equivalente a casi 2 horas. Por lo tanto, la evaluación de una base de datos

completa puede tardar días o semanas, lo que no es eficiente al tener en cuenta los tiempos asignados para la realización de las pruebas en una auditoría promedio.

A pesar de la importancia de las pruebas de auditoría, estas por si solas no pueden garantizar la seguridad de la información en una organización (W. G. J. Halfond et al., 2008; ISO/IEC, 2015). Las organizaciones deben implementar un sistema de gestión de seguridad de la información, que se encargue de mitigar los riesgos de seguridad (ISO/IEC, 2013). Dentro de este sistema de gestión, las auditorías juegan un papel importante. Las organizaciones definen un programa de auditorías, asignando tiempos para la realización de cada una (ISO/IEC, 2007). Generalmente, los tiempos asignados a un auditor para evaluar un riesgo de seguridad como las inyecciones sobre los ambientes web es limitado. Durante este tiempo, se espera que se apliquen las pruebas de auditoría, no solo para evaluar el riesgo de inyección, sino todas las demás vulnerabilidades web. Aunque la evaluación del riesgo mencionado se realiza de manera automatizada, con un tiempo de evaluación extenso la prueba puede verse afectada por la premura del proceso de auditoría o hasta por cortes de energía y fallas de red. Si esto ocurre, una prueba de penetración no exitosa carece de confiabilidad (ISO/IEC, 2009).

Con el objetivo de realizar las pruebas de penetración sobre el riesgo de inyecciones SQL inferenciales, un auditor de sistemas o profesional de seguridad de la información puede usar los siguientes algoritmos: búsqueda secuencial basada en tiempo, búsqueda binaria basada en error, búsqueda binaria con expresiones regulares, búsqueda binaria con diccionarios, búsqueda binaria probabilística, extracción bit a bit (Slaviero, 2012) y extracción de múltiples bits usando inyecciones de tiempo (documentado teóricamente pero sin implementación real) (Redwood, 2016). Estos algoritmos hacen parte de lo que, en el campo de la auditoría, se denominan técnicas y herramientas de auditoría asistidas por computador (CAATTs, por sus siglas en inglés), las cuales facilitan la prueba de penetración. Sin embargo, de acuerdo con la literatura encontrada, los tiempos siguen siendo significativos (Štampar, 2016).

1.2 Pregunta de Investigación

A partir de lo anterior, la pregunta de investigación de esta tesis es la siguiente:

¿Cómo reducir el tiempo que se emplea en las pruebas de penetración para evaluar los riesgos de inyecciones SQL inferenciales en un proceso de auditoría de tecnologías de la información?

1.3 Objetivo

1.3.1 General

Reducir el tiempo usado en las pruebas de penetración informática durante la evaluación del riesgo de inyecciones SQL inferenciales existente en las aplicaciones empresariales bajo ambiente web.

1.3.2 Específicos

- Realizar un análisis comparativo de los algoritmos, técnicas y herramientas usadas para llevar a cabo la evaluación del riesgo de inyecciones SQL inferenciales.
- Implementar una herramienta de auditoría para la evaluación del riesgo de inyecciones SQL inferenciales con base en el análisis de los algoritmos realizados que permita disminuir el tiempo de las pruebas.
- Validar el uso de la herramienta diseñada en la evaluación del riesgo Inyecciones SQL Inferenciales en sitios web corporativos.

1.4 Justificación

Garantizar la seguridad de la información no solo es una necesidad sino también una obligación (Ley estatutaria 1581 de 2012). Debido a esto, la organización internacional de estándares generó uno internacional (ISO/IEC 27001) que fue acogido como norma técnica colombiana por el organismo nacional de normalización de Colombia (ICONTEC), el cuál sirve de guía para la implementación de un sistema de gestión de seguridad de la información (ISO/IEC, 2013).

Esta norma, establece la necesidad de identificar riesgos de seguridad para poder establecer controles que permitan su mitigación. Los riesgos que no se identifiquen y evalúen correctamente, podrán ser explotados de manera malintencionada afectando la disponibilidad, integridad y confidencialidad de la información, los cuáles son los pilares de su seguridad (Gollmann, 1999).

Con el objetivo de mejorar la eficiencia en la identificación y valoración de riesgos, la misma organización internacional generó unas directrices en las que se incluye la necesidad de realizar un proceso de identificación de vulnerabilidades. Este proceso incluye la realización de pruebas de penetración manuales, herramientas automáticas de explotación de vulnerabilidades y evaluaciones de seguridad por parte de profesionales (ISO/IEC, 2009).

Esta tesis se enfoca en mejorar este proceso de identificación y evaluación de vulnerabilidades descrito por esta norma, la cual indica “que las herramientas y técnicas de penetración pueden dar resultados falsos, a menos que la vulnerabilidad sea explotada exitosamente” (ISO/IEC, 2009, p. 55). El mejorar este proceso impacta positivamente sobre la identificación de vulnerabilidades y, de manera indirecta, en su mitigación a través del sistema de gestión.

Ahora bien, aunque de acuerdo con OWASP existen cientos de vulnerabilidades web (OWASP, 2017), la denominada inyección ha sido la que presenta mayores niveles de criticidad de manera consecutiva en los últimos años considerando que en el resultado de la última evaluación realizada en el 2013 por esta misma organización, esta vulnerabilidad se encuentra en el mismo lugar (OWASP, 2013).

Una inyección ocurre cuándo se ingresan datos no confiables sin validar a un intérprete como parte de una consulta o comandos(OWASP, 2017). De todos los tipos de inyecciones, las inyecciones SQL, No-SQL y, más recientemente, inyecciones ORM, son críticas debido a que son interpretes ligados a la base de datos y un atacante puede afectar directamente la disponibilidad, integridad y confidencialidad de la información almacenada. De estos tipos, las inyecciones SQL son las que llevan más tiempo presentándose en las aplicaciones web debido a que la mayoría de estas usan estos motores de base de datos. Por esa razón, el mejorar el proceso de identificación y evaluación de la vulnerabilidad a inyecciones SQL permitirá que las organizaciones puedan reducir el tiempo en las pruebas realizadas sobre el mayor riesgo de seguridad en sus ambientes web.

De acuerdo con la clasificación más citada (W. G. J. Halfond et al., 2008), las inyecciones SQL pueden ser:

1. Tautológicas
2. Consultas Ilegales/Lógicamente Incorrectas
3. Uniones
4. Consultas a cuestas
5. Procedimientos almacenados
6. Inferenciales
7. Codificación Alternativa

La identificación de esta vulnerabilidad consiste en evaluar las entradas de cada página web perteneciente al entorno de la aplicación y ejecutar una prueba de penetración con el fin de identificar si la entrada es vulnerable, y si lo es, identificar la afectación que se puede causar a la información (Clarke & Alvarez, 2009). En caso de ser posible, la extracción de la información permite evaluar la criticidad del riesgo y establecer controles para evitar la pérdida de confidencialidad (Štampar, 2016).

A diferencia de los otros tipos de inyecciones SQL, la prueba de penetración sobre el tipo de inyecciones SQL inferenciales consiste en la extracción carácter por carácter de la información almacenada en la base de datos. Dependiendo del tamaño de la misma, esta prueba puede durar días o incluso semanas (Štampar, 2016).

De acuerdo con los estándares internacionales, una auditoría “se lleva a cabo durante un periodo de tiempo delimitado y con recursos finitos” (ISO/IEC, 2007, p. 5). Este tiempo, generalmente dado por el programa de auditoría y al presupuesto de la organización, toma

en cuenta no solo el proceso mismo de auditoría, sino también el tiempo y los costos asociados al desplazamiento y alojamiento de los auditores (ISO/IEC, 2007). Por lo anterior, con el fin de optimizar el costo, los tiempos para la realización de la auditoría son ajustados. Un auditor de seguridad en ambientes web debe realizar sus pruebas de manera exhaustiva, con el fin de identificar con mayor precisión las vulnerabilidades web, y, además, realizarlas en el menor tiempo posible con el fin de cumplir con el cronograma de auditoría y no incrementar el costo de la misma. No es menos importante indicar que una prueba de auditoría de días o semanas, es más susceptible a fallas de hardware, red o eléctricas.

En vista de lo anterior, cualquier intento por reducir el tiempo de realización de las pruebas de auditoría para la identificación y evaluación del sub-tipo de inyección SQL con mayor complejidad tendrá un impacto directo en facilitar a las organizaciones la generación de planes de acción que mitiguen la vulnerabilidad más crítica en los ambientes web. La complejidad de las inyecciones SQL inferenciales se agudiza sobre todo en las que son basadas en tiempo. Para realizar la evaluación de esta vulnerabilidad, se debe escoger un tiempo de retraso a inyectar en cada consulta realizada sobre la base de datos. Este tiempo no puede ser menor debido a que puede confundirse con el tiempo de carga de la página. Pero si se coloca un tiempo mayor, como 30 segundos o más, el proceso se vuelve ineficiente y se corre el riesgo de activar las “excepciones por exceso de tiempo incluidas en la base de datos o en el entorno de trabajo de la aplicación web” (Slaviero, 2012, p. 258). Por esa razón, la evaluación de estas inyecciones SQL son “entre 10 a 2 mil veces” más lentas que la evaluación de otros tiempos de inyecciones (Štampar, 2016, p. 323).

1.5 Alcance

Existen 7 tipos de inyecciones SQL: Tautológico, Consultas Ilegales/Lógicamente Incorrectas, Uniones, Consultas a cuestas, Procedimientos almacenados, Inferenciales y Codificación Alternativa (W. G. J. Halfond et al., 2008). Cada uno de estos tipos tienen algoritmos diseñados para su evaluación a través de pruebas de penetración. El foco de este proyecto de investigación son las inyecciones SQL inferenciales y está delimitado por la cantidad y tipo de algoritmos usados en estas inyecciones que son objeto de análisis.

Además, está limitado por las variables que se pueden usar para la evaluación de estos algoritmos.

En relación con la cantidad y tipo de algoritmos existentes usados en inyecciones SQL inferenciales, se analizarán y compararán aquellas que se documentaron e implementaron hasta el 31 de diciembre de 2016. Asimismo, las herramientas que se analizarán serán aquellas que cuenten con una implementación con licenciamiento libre o con una versión demo hasta la fecha antes mencionada.

Con respecto a las técnicas de evaluación, se analizarán las inyecciones SQL no basadas en tiempo y basadas en tiempo. Estas últimas limitadas a las inyecciones que hacen uso de las funciones de base de datos para inyectar un retardo de tiempo.

Y en relación con las variables a medir en el proceso de evaluación, se evaluará la eficiencia del algoritmo a partir del tiempo de ejecución de la evaluación del riesgo. No se analizarán otras variables cómo la cantidad de ruido generado ni el uso de los recursos de la máquina.

1.6 Metodología

La metodología para responder la pregunta de investigación, utiliza un enfoque mixto con el fin de conseguir los objetivos formulados: El análisis de los algoritmos y la comparación de los mismos se basan en un análisis cualitativo y para la evaluación empírica de los algoritmos y herramientas, y en general, de los otros componentes de esta investigación, se usará un enfoque cualitativo por lo que existen datos recolectados no de manera estándar y sirven para afinar la pregunta de investigación (Hernández, Fernández, & Baptista, 2014). Adicionalmente, para el cumplimiento de los objetivos nos basamos en la hermenéutica para la interpretación de las pruebas y comparaciones realizadas.

Para esto, se realizarán las siguientes fases:

1. Fase Exploratoria:
    a. Realizar una revisión de la literatura existente sobre los siguientes temas:
        i. Seguridad de la información y ciberseguridad.
        ii. Ambientes web.

iii. Inyecciones SQL.
iv. Pruebas de penetración y análisis de seguridad.
v. Investigaciones recientes sobre inyecciones SQL inferenciales.

2. Fase de Análisis:
    a. Analizar los algoritmos desarrollados por la comunidad académica y profesional para ejecutar pruebas de penetración de inyecciones SQL usando la metodología *enfoque científico para el análisis de algoritmos* (Robert Sedgewick & Philippe Flajolet, 1996).
    b. Creación de un entorno de prueba: En esta actividad se crearán dos bases de datos en MySQL y SQL Server, una alimentada con bytes aleatorios y otra base de datos con información de palabras reales en inglés y en español.
    c. Analizar el tiempo tomado por las herramientas para la evaluación del riesgo de inyecciones SQL Inferenciales y definir la mejor en términos de tiempo de extracción.
3. Fase de diseño e implementación:
    a. Diseñar la herramienta de acuerdo a la metodología, recomendaciones y técnicas para diseño de herramientas automatizadas definidas por el Object Management Group OMG en la metodología UML 2 (The Object Management Group, 2005).
    b. Realizar la implementación usando el algoritmo más eficiente de acuerdo al análisis de la fase 2.
4. Fase de validación:
    a. Realizar la validación en un entorno corporativo con la herramienta desarrollada en la fase 3. Para esto, se elabora un programa de auditoría de acuerdo a la metodología de ISACA (ISACA, 2016).
    b. Analizar los resultados y elaborar el documento final.

Siguiendo la metodología desarrollada, se podrán cumplir los objetivos definidos. Y se resolverá la pregunta de investigación.

2. Inyecciones SQL Inferenciales

2.1 Seguridad y Ciberseguridad de la información

El comité de seguridad nacional de sistemas de Estados Unidos (CNSS) define seguridad de la información cómo “la protección de la información y sus elementos críticos, incluyendo los sistemas y hardware que usan, almacenan, y transmiten esa información” (Whitman & Mattord, 2012, p. 8). Los pilares de la seguridad de la información son confidencialidad, integridad, y disponibilidad (Gollmann, 1999) llamado también el triángulo CIA o tríada CIA, por sus siglas en inglés. Asimismo, la organización de estandarización internacional (ISO) indica que el contexto de información incluye la que es impresa o escrita en papel, almacenada electrónicamente, transmitida por correo o por medios electrónicos, la mostrada en películas, transmitida en conversaciones, entre otras (ISO/IEC, 2013).

Existe un sub-componente de la seguridad de la información llamado Seguridad de la Tecnología de la Información y la Comunicación (en adelante seguridad ICT por sus siglas en inglés) “la cual se encarga de la protección de los sistemas basados en la tecnología real en la que se almacena la información y / o se transmite comúnmente” (von Solms & van Niekerk, 2013). De acuerdo con la organización de estándares, la seguridad ICT incluye todos los aspectos relativos a la “definición, logro y mantenimiento de la confidencialidad, integridad, disponibilidad, no repudio, la responsabilidad, la autenticidad y fiabilidad de los recursos de información” (ISO/IEC, 2004, p. 3).

Esta definición de la seguridad ICT es muy similar a la definición de seguridad de la información, aunque limitados al aspecto tecnológico y puede notarse que tiene más conceptos que la tríada. Estos conceptos hacen parte de lo denominado “seguridad de los datos” (von Solms & van Niekerk, 2013, p. 98).

El concepto de seguridad ICT está relacionado con el concepto de ciberseguridad. Este término cuenta con 8 o más definiciones. Después de analizar 5 definiciones relevantes, los siguientes autores dan la siguiente definición:

“Ciberseguridad es la organización y colección de recursos, procesos y estructuras usadas para proteger el ciberespacio (cyberspace) y los sistemas que habilitan el ciberespacio de las ocurrencias que desalinean tanto los derechos de propiedad percibidos (de jure) cómo los reales (de facto)” (Craigen, Diakun-Thibault, & Purse, 2014, p. 17).

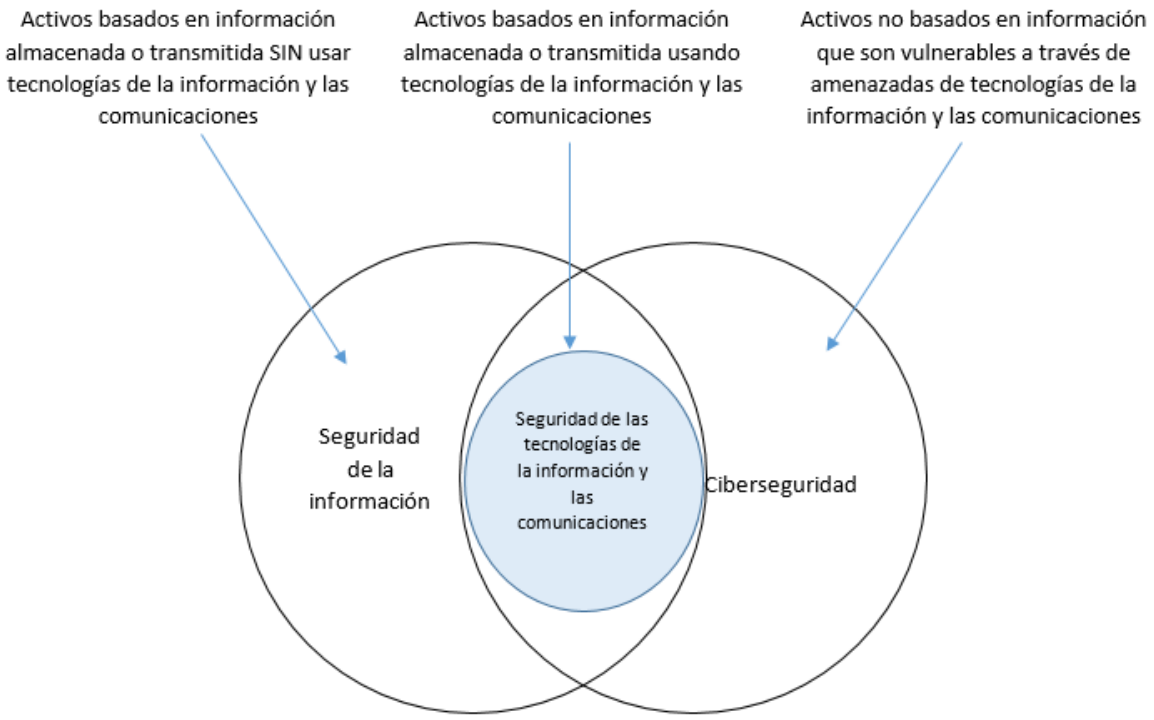
La anterior definición está de acuerdo con la ofrecida por la ISO/IEC 27032 (ISO/IEC, 2012).

La Unión Internacional de Telecomunicaciones (International Telecommunication Union, ITU) define ciberseguridad cómo:

> *La colección de herramientas, políticas, conceptos de seguridad, medidas de seguridad, directrices, enfoques de gestión de riesgos, acciones de formación, mejores prácticas, aseguramiento y nuevas tecnologías que se pueden utilizar para proteger el medio ambiente y la organización cibernética y los bienes de los usuarios. Organización y los activos de los usuarios son los dispositivos informáticos, personal, infraestructura, aplicaciones, servicios, sistemas de telecomunicaciones, y la totalidad de la información transmitida y / o almacenada en el entorno cibernético conectados. La ciberseguridad garantiza la consecución y el mantenimiento de las propiedades de seguridad de la organización y los activos de los usuarios contra los riesgos de seguridad relevantes en el entorno cibernético. Los objetivos generales de seguridad comprenden lo siguiente: disponibilidad, integridad, que puede incluir la autenticidad y el no repudio, confidencialidad* (ITU, 2015, p. 97).

Para la ISO/IEC 27032, ciberseguridad es un término que agrupa las brechas entre los conceptos de seguridad de la información, seguridad de Internet, seguridad de redes y seguridad ICT (ISO/IEC, 2012). A pesar de la similitud en las definiciones, las diferencias entre estas pueden verse en la Figura 1.

Figura 1 *Relaciones entre seguridad de la información y comunicación, seguridad información, y ciberseguridad.*



Fuente: Adaptado de (von Solms & van Niekerk, 2013)

La diferencia entre seguridad ICT y ciberseguridad radica en que la primera tiene su foco en la información y el efecto que se produce en la misma mientras que el término ciberseguridad incluye esto mismo y además, el efecto que pueda traer sobre las personas, organizaciones, gobiernos, entre otros (von Solms & van Niekerk, 2013). Para explicar esta diferencia, estos autores colocan algunos ejemplos, los cuales no se tratan de incidentes de seguridad de la información, pero sí de ciberseguridad. Uno de estos ejemplos es el acoso virtual, donde se usan medios tecnológicos para la transmisión de información que afectan personas, empresas u organizaciones sin afectar la información misma. Aunque la información no se afecta (seguridad de la información), si lo hacen los activos que la utilizan (ciberseguridad).

Aunque no exista un concepto único, este proyecto de tesis se sustenta en la definición de la ciberseguridad cómo la protección del ciberespacio, incluyendo la infraestructura que lo habilita (seguridad ICT) y las personas, organizaciones, gobiernos que lo acceden

(ciberseguridad de acuerdo con von Solms & van Niekerk, 2013). Igualmente se acoge a la definición de ciberespacio cómo un mundo artificial infinito donde los humanos navegan en un espacio basado en la información (Benedikt, 1992).

Este concepto de ciberseguridad es el que enmarca el área de acción de este proyecto de investigación. Su objetivo principal es reducir el tiempo empleado en las pruebas de penetración con el fin de mejorar el proceso de auditoría que permita mantener la información segura.

2.2 Ambientes web

En (Berners-Lee, 1992), se presentó el diseño World-Wide Web (W3), un sistema de información global compuesto por protocolos y formatos de datos estándar que permitía que varios dispositivos usaran un software local llamado navegador y accedieran a información (en formato HTML) publicada en servidores usando un protocolo llamado HTTP. Para finales de los años 90's, en los países industrializados, la W3 ya era el principal medio para diseminar información (Florescu, Levy, & Mendelzon, 1998).

Los ambientes web son todos aquellos que usan la arquitectura de la W3 y se basan en el lenguaje de marcas de hipertexto (HyperText Markup Language, HTML) el cual permite la creación de páginas web (ISO/IEC, 2000). Aunque este lenguaje se usa para mostrar información en aplicaciones de escritorio que lo interpretan, conocidos cómo navegadores, no almacenan información de manera dinámica. Para este fin, se usan motores de base de datos (Florescu et al., 1998).

Aunque existen varios tipos de base de datos, esta tesis de maestría analiza el riesgo de la información almacenada sobre bases de datos relacionales, las cuales usan el lenguaje de consultas estructurado (Structured Query Language, SQL) para el acceso a los datos (Nayak, Poriya, & Poojary, 2013). Aunque el lenguaje SQL fue primero un estándar ANSI y ahora es un estándar ISO (ISO/IEC, 2011), existen variantes del lenguaje implementados en los principales motores de base de datos. Los datos organizacionales se almacenan en

estos motores y son accedidos por un ambiente web que muestra la información a través de los protocolos indicados anteriormente.

Para asegurar esta información, es necesario incorporar mecanismos de seguridad de la información, que permitan garantizar adecuados niveles de confidencialidad, integridad y disponibilidad. Esta investigación se limita a la información almacenada en bases de datos SQL y que son accedidas a través de ambientes web.

2.3 Inyecciones SQL

Los riesgos de seguridad en ambientes web han sido objeto de estudio por la comunidad académica y profesional. Una de las iniciativas que más ha profundizado en este aspecto es el proyecto abierto de seguridad en aplicaciones web (en adelante OWASP por sus siglas en inglés), la cual periódicamente evalúa los principales riesgos de seguridad que una organización debe prevenir (OWASP, 2017). Esa evaluación la realiza a través de una metodología propia llamada OWASP Risk Rating Methodology enfocada solo en riesgos de seguridad y basada en los criterios mostrados en la figura 2.

Figura 2 *Criterios de evaluación OWASP Risk Rating Methodology*.

Agentes Amenazados	Vectores de ataque	Prevalencia de la debilidad	Detectabilidad de la debilidad	Impactos técnicos	Impacto sobre el negocio
Aplicación especifica	Fácil	Extendida	Fácil	Severos	Aplicación / Negocio específico
	Promedio	Común	Promedio	Moderados	
	Difícil	Poco común	Difícil	Menores	

Fuente: Adaptado de (OWASP, 2017)

Aunque de acuerdo con (OWASP, 2017) existen cientos de vulnerabilidades web, la denominada inyecciones es la que ha presentado mayor criticidad por varios años consecutivos aplicando la metodología mostrada en la figura 2. Las inyecciones “ocurren cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta. El atacante puede inducir engañosamente al intérprete para ejecutar comandos no intencionados o permitir acceder a datos sin la propia autorización” (OWASP, 2017, p. 6). Para que el intérprete tome cómo válido los datos malintencionados, se introducen a

través de campos de entrada código no autorizado para ser ejecutado por la aplicación. De la definición dada por OWASP se desprende que todos los lenguajes interpretados son vulnerables a las inyecciones.

De todos los tipos de inyecciones, las de código SQL, No-SQL y, más recientemente, ORM, son inyecciones críticas debido a que los intérpretes están ligados a la base de datos y un atacante puede afectar directamente la disponibilidad, integridad y confidencialidad de la información almacenada. De los intérpretes antes mencionados, los de SQL son los que son objeto de estudio en esta investigación.

De acuerdo con (W. G. J. Halfond et al., 2008), existen siete tipos de inyecciones SQL:

1. Tautológico
2. Consultas Ilegales/Lógicamente Incorrectas
3. Uniones
4. Consultas a cuestas
5. Procedimientos almacenados
6. Inferenciales
7. Codificación Alternativa

Para ver una descripción breve de cada una, en la sección de investigaciones recientes, se encuentra la última definición disponible de cada una, específicamente en la tabla 2.

De todos los tipos de inyecciones SQL, las denominadas inferenciales, presentan una mayor complejidad para su evaluación. Aunque el riesgo de inyecciones SQL inferenciales es llamada por algunos autores como inyección SQL Ciega (Slaviero, 2012), esta tesis se focaliza en este tipo y usa el nombre Inferencial ya que describe mejor el riesgo al basarse en el proceso llevado a cabo para realizar la extracción de información, apoyando así las definiciones realizadas por otros autores (W. G. J. Halfond et al., 2008; Štampar, 2016).

Al igual que los otros tipos de inyecciones SQL, la evaluación de esta vulnerabilidad incluye la extracción de datos y del esquema de la base de datos (W. G. J. Halfond et al., 2008). Para realizar la evaluación de esta vulnerabilidad, existen dos técnicas conocidas:

Inyecciones Ciegas o Ataques de Tiempo o como es llamado en (Štampar, 2016): Basadas en Tiempo y No basadas en tiempo, una definición más descriptiva.

Las inyecciones basadas en tiempo inducen a la base de datos a esperar un tiempo determinado por el atacante. Esto se puede hacer usando las funciones propias de cada base de datos cómo SLEEP (en Mysql), PG_SLEEP (en Postgres), WAIT FOR DELAY (en SQL Sever), DBMS_LOCK.SLEEP (en Oracle) (Clarke & Alvarez, 2009) o forzando la base de datos a hacer consultas pesadas que consuman más tiempo del habitual (Alonso, Daniel Kachakil, Bordón, Guzmán, & Beltrán, 2007).

Las inyecciones no basadas en tiempo analizan los cambios en la presentación del ambiente web. Por ejemplo, si la inyección SQL enviada contiene un error en tiempo de ejecución y este error es mostrado en la página web, entonces este comportamiento puede usarse para un criterio de falso y verdadero.

En ambas técnicas es necesario “booleanizar” la inyección: cada consulta realiza una pregunta y de acuerdo a la técnica se infiere si es verdadero o falso. Ejemplo: ¿El primer carácter del nombre de la base de datos es la letra A? Si la respuesta es un cambio en la presentación de la página web o un tiempo de carga más demorado de lo normal, se puede inferir cómo verdadero. De esa manera puede extraerse toda la base de datos, aunque tomando más tiempo que con las otras técnicas (Štampar, 2016). En la sección 3 se analizará profundamente este tipo de inyección SQL.

Esta investigación se enfoca en el riesgo de inyecciones SQL inferenciales. Se toman los tipos basados en tiempo y no basados en tiempo. Sobre los primeros, se usan las funciones propias para el retraso de tiempo de acuerdo al motor de base de datos y no se analizan las técnicas que usan consultas pesadas.

2.4 Pruebas de penetración y análisis de seguridad

De acuerdo con la norma ISO/IEC 27002, deben realizarse pruebas de penetración para encontrar vulnerabilidades de seguridad con el objetivo de establecer controles que las mitiguen (ISO/IEC, 2015). Para el caso de los riesgos asociados a las inyecciones SQL, estos controles se clasifican en: prácticas defensivas de código y otras técnicas de

prevención y detección (W. Halfond & Orso, 2007). Dentro de las prácticas defensivas de código se encuentran (W. G. J. Halfond et al., 2008):

- Verificación de cada entrada del entorno
- Codificación (Encoding) de dichas entradas
- Validación a través de la coincidencia positiva de patrones
- Identificación de todas las entradas del entorno web.

En otras técnicas de prevención y detección encontramos (W. G. J. Halfond et al., 2008):

- Pruebas de caja negra.
- Verificadores de código estático.
- Análisis combinado estático y dinámico.
- Enfoques basados en análisis de entradas contaminadas.
- Paradigmas de desarrollo con nuevas consultas.
- Sistemas de detección de intrusos.
- Filtros de proxy.
- Conjunto de instrucciones aleatorio.

La primera técnica de prevención, llamada prueba de caja negra es la prueba de penetración a la que hace referencia la norma ISO 27002.

Las pruebas de penetración en sistemas de información iniciaron en el año 1985, basado en las ideas de (Weissman, 1985) quien escribió un ensayo de 10 secciones donde describe un método integral para la realización de análisis de penetración, dentro del cual las pruebas de penetración eran un componente en su metodología. Estas ideas fueron estructurándose y aplicándose a sistemas de información, cuando se escribió el artículo: “enfoque sistemático para la prueba de penetración de un sistema de software existente” (Pfleeger, Lawrence, & Theofanos, 1989, p. 613). En lo definido por estos autores, dicha prueba inicia con un análisis del sistema, hipótesis de posibles fallas, confirmación o rechazo de la hipótesis, y extensión de una hipótesis confirmada. En ese enfoque se utiliza el análisis de los requisitos y controles de seguridad para identificar los objetos que se van

a proteger y para anticipar posibles ataques a esos objetos. Posteriormente, (Farmer & Venema, 1993) indicaron que se podía mejorar la seguridad de un sitio rompiéndolo o en otras palabras, realizar una auditoría simulando ser un intruso del sistema, lo cual confirmaba los dos estudios anteriores. Esta técnica es conocida ahora como pruebas de penetración. En el año 1998, (Krsul, 1998) propuso en una tesis doctoral una metodología de análisis de vulnerabilidad del software. Una idea similar fue propuesta por (Du, 1998) con el nombre de prueba de vulnerabilidad. Desde el año 1999, el análisis de vulnerabilidades y las pruebas de penetración son unidas para tener un mecanismo de auditoría denominado Análisis de vulnerabilidad y pruebas de penetración (Vulnerability Analysis and Penetration Testing VAPT) (Jansen, Walsh, Dolan, Wright, & Montequin, 2000).

Este análisis de Vulnerabilidad y Pruebas de Penetración es un análisis sistemático usado para la evaluación de la seguridad de la información para identificar y -reportar riesgos de seguridad. El análisis de vulnerabilidad "implica el descubrimiento de un subconjunto de entrada con el que un usuario malintencionado puede explotar un sistema para conducirlo a un estado inseguro […]. El proceso general consiste en escanear de manera activa el sistema para [encontrar] las […] fallas de software" (Shah & Mehtre, 2015, p. 1). El paso siguiente es realizar pruebas de penetración, el cual es un método exhaustivo para poner a prueba el sistema y determinar la dificultad que tiene un atacante para penetrar los controles de seguridad establecidos por la organización para prevenir los riesgos de accesos no autorizados a la información (Shah & Mehtre, 2015).

Para realizar las pruebas de penetración y agilizar los resultados de la auditoría en ciberseguridad se pueden aplicar los mismos conceptos que usan las técnicas y herramientas de auditoría asistidas por computador (Valencia Duque & Tamayo Arias, 2016).

La Norma Internacional de Auditoría (NIA, 2002a, 2002b) y la Asociación de Auditoría y Control de Sistemas de Información (ISACA, 2008) definen CAATTs, como sistemas automatizados que el auditor usa como parte de los procedimientos de auditoría. Los beneficios de usar estas herramientas son la optimización del tiempo por parte de los auditores y la revisión exhaustiva que no podría realizarse de manera manual. Herramientas con principios similares a estos son usadas ampliamente tanto en el análisis de vulnerabilidad y las pruebas de penetración (Shah & Mehtre, 2015). Durante esta

investigación, se analizaron las herramientas de auditoría usadas para las pruebas de penetración sobre el riesgo de inyecciones SQL inferenciales y se determinará si es posible realizar una mejora a estas herramientas o la construcción de una herramienta más eficiente.

Esta investigación se concentra en la reducción del tiempo que se emplea en las pruebas de penetración y toma los conceptos de prueba de caja negra. En sus objetivos se encuentra el análisis de las herramientas automatizadas y, precisamente, uno de sus resultados es una herramienta para una auditoría asistida por computador.

2.5 Investigaciones recientes sobre inyecciones SQL inferenciales

Debido al aumento de la delincuencia informática y a la amplia utilización de los sitios web que almacenan todo tipo de información disponible a través de Internet, se creó la Ley 1266 de 2008 o Ley de habeas data garantizando la necesidad de mantener la información personal con el derecho a la intimidad (Ley estatutaria 1266, 2008). Por lo anterior, toda organización, empresa o persona que tenga almacenada información en bases de datos está obligada por el estado a proteger sus entornos web.

Se han definido varias metodologías para realizar el análisis de vulnerabilidad y pruebas de penetración (Shah & Mehtre, 2015), las cuales se usan con el objetivo de encontrar riesgos y evaluar los controles implementados en los componentes de la infraestructura tecnológica (G. Stoneburner, Goguen, & Feringa, 2002). Además, se realizan congresos de seguridad con el fin exclusivo de encontrar vulnerabilidades nuevas (llamadas Zero Day), mostrar resultados de la aplicación de VAPT en entornos reales (Tian, Yang, Xu, & Si, 2012) y difundir nuevas herramientas de auditoría. Por tanto, se ha producido un ecosistema donde entusiastas y profesionales de la ciberseguridad ayudan a mejorar los entornos web que todos usamos y una amplia cantidad de herramientas a disposición de los auditores. Este ecosistema se basa en la programación de eventos con conferencias que pueden durar varios días, cómo las programadas por OWASP (OWASP, 2016) y BSIDES (BSIDES, 2016) con capítulos en varias ciudades a nivel mundial. Además, está

la conferencia más concurrida a nivel mundial llamada DEF CON, que reúne a más de 15 mil personas interesadas en la ciberseguridad (DEF CON, 2016). A nivel colombiano, se realiza la conferencia BSIDES del capítulo Colombia (Bsides, 2016) y la conferencia DragonJAR (DragonJar, 2016).

Las pruebas de penetración para la evaluación del riesgo de inyecciones SQL inferenciales se realizaban de manera manual, tal y como lo describe quien identificó por primera vez este subtipo del riesgo de inyecciones SQL (Anley, 2002a). Ese mismo año, el mismo autor amplía su investigación sobre dicha vulnerabilidad y muestra el que sería el primer algoritmo para la prueba de penetración: extracción bit a bit (Anley, 2002b). A partir de ese momento, se empezaron a realizar investigaciones encaminadas a su automatización y optimización. Ese mismo año (Cerrudo, 2002) presenta en una conferencia de ciberseguridad llamada Black Hat su investigación sobre este riesgo, mostrando cómo afecta la integridad no solo de la base de datos, sino de la infraestructura donde se encuentra. En (Maor & Shulman, 2003) se presenta un nueva forma de penetración a través de la inyección de errores además de otras técnicas hasta ese momento desconocidas para la extracción de información que permitían hacer consultas más avanzadas.

Teniendo en cuenta la complejidad manual de las pruebas de evaluación, (Cameron Hotchkies, 2004) desarrolla una herramienta denominada SQueal (ahora llamado Absinthe), la cual implementa un nuevo algoritmo para la prueba de penetración: algoritmo de búsqueda binaria. Al presentar este algoritmo, explica que es una optimización de otro el cual consiste en una búsqueda secuencial. Hasta ese momento, la inyección inferencial se conocía cómo la inyección ciega (algunos autores continúan usando ese nombre) hasta que (Litchfield, 2005, p. 4) indicó la razón por la cual debería llamarse inferencia: “No se transmiten datos, pero observando diferencias en la respuesta de la aplicación, el atacante puede inferir el valor de los datos”. Además, el mismo autor presenta ejemplos de cómo usar este riesgo para afectar la confidencialidad a través de minería datos. Posteriormente se realiza una investigación (Alonso et al., 2007) donde se demuestra que puede explotarse este riesgo aún si la base de datos no tiene métodos para inyección de tiempo, usando consultas pesadas. Este autor presenta una nueva herramienta llamada Marathon Tool para la base de datos SQL Server. El año siguiente, (W. G. J. Halfond et al., 2008) escribe el que sería el artículo más citado sobre el tema de inyecciones SQL, realizando

una taxonomía del riesgo y clasificando el riesgo de inyecciones SQL Inferenciales en dos: Inyección ciega (no basada en tiempo) y ataques de tiempo. En (Halde, 2008) se presenta varias herramientas existentes y mejora una llamada SQLiX, la cual compara con Acunetix, SQLmap, wapití, paros y pixy. Durante el año 2009, se escribe uno de los libros más citados sobre el tema (Clarke & Alvarez, 2009) que recopila las técnicas y algoritmos documentados hasta ese momento. El capítulo 5 de este libro es dedicado al riesgo de inyecciones SQL ciegas y explica los siguientes algoritmos para las pruebas de penetración: algoritmo secuencial, el algoritmo de búsqueda binaria y el algoritmo bit a bit. Además, muestra cómo usar los retardos de tiempos y aplicar en diferentes bases de datos dichos algoritmos. Dentro de las herramientas, este libro muestra Absinthe, BSQL Hacker, SQLBrute, SQLNinja y Squeeza. No obstante, existen otros algoritmos que no se analizan en el libro ni se comparan entre sí.

A partir de ese momento, las investigaciones se encaminan en mejorar los algoritmos, diseñar nuevos y generar una herramienta para cada algoritmo encontrado. La investigación de (Evteev, 2010) aplica los algoritmos con la técnica inyección de errores (no basada en tiempo) y (Quatrini & Rondini, 2011) crean una optimización de los algoritmos de extracción a través de expresiones regulares. En (Toews & Scott Behrens, 2012) los autores presentan una nueva herramienta, llamada BBQSQL y una optimización al algoritmo secuencial y búsqueda binaria ordenando los caracteres por su probabilidad de ocurrencia, a la que se llama en esta tesis Búsqueda binaria/secuencial optimizada por probabilidad.

En (Nagpal, Singh, Chauhan, & Panesar, 2015) los autores presentan una comparación entre las siguientes herramientas de extracción: Havij, BSQL Hacker, Marathon Tool y SQL Power Injector. También nombran a SQL Poizon, aunque no le realizan ningún análisis. En la tesis de Maestría de (Wheeler, 2015) explica una herramienta llamada BlindCanSeeQL que extrae información a través de otra optimización de un algoritmo de búsqueda usando diccionario, pero no se cuenta con un lanzamiento de la aplicación para uso general. Durante el año 2016 se publicó una tesis (Hudák, 2016) donde se genera una estrategia de prevención y se evalúa con la herramienta SQLMAP. Además, en este mismo año, uno de los creadores de la herramienta SQLMAP (Štampar, 2016) realiza una investigación comparando empíricamente tres algoritmos optimizados. En la tabla 1 se muestran los tres

algoritmos analizados (más sus optimizaciones) y el tiempo usado en la extracción de información.

En ese mismo año, PhD. Owen Redwood publica en su blog (Redwood, 2016) un algoritmo al que llama extracción de múltiples bits por solicitud a partir de la vulnerabilidad inyección SQL completamente ciega, aunque sin ninguna implementación práctica. Igualmente, en el 2016 se realizó una clasificación diferente de la detección y prevención de ataques de inyecciones SQL (Aliero, Ardo, & Ghani, 2016). Además, en este mismo año se realizó una revisión sistemática de literatura analizando 20 diferentes artículos sobre las inyecciones SQL(Lawal, Sultan, & Shakiru, 2016) que se usó para esta investigación y se publicaron otros artículos sobre cómo realizar aplicaciones que se defiendan de los ataques basándose en la intención del usuario (Chenyu & Fan, 2016) y en la arquitectura de la consulta(George & Jacob, 2016).

Tabla 1 *Comparación de algoritmos de búsqueda*

Método	# de solicitudes	Inyección ciega (s)	Ataques de tiempo (s)
Búsqueda secuencial (regular)	5412	305.44 / 800.27	367.13 / 882.84
Búsqueda secuencial (optimizada)	2140	120.67 / 293.11	184.97 / 414.58
Búsqueda binaria (regular)	537	30.34 / 72.17	241.89 / 517.75
Búsqueda binaria (optimizada)	494	27.89 / 64.63	173.57 / 375.95
Extracción bit a bit (regular)	537	30.25 / 68.74	315.41 / 487.24
Extracción bit a bit (optimizada)	470	26.63 / 59.21	238.19 / 485.72

Fuente: (Štampar, 2016)

Durante el año 2017, se realizaron varios estudios dedicados a esta vulnerabilidad. En (Ghafarian, 2017), se propone un método hibrido para la prevención de los ataques de inyecciones SQL. Este método consiste en una capa estática de base de datos y otra dinámica de CGI, la cual consiste en tres partes:

1. Modificar las tablas de la base de datos para agregar un registro que tenga el símbolo pesos.

2. Ejecutar un CGI que analice la entrada que se enviará a la base de datos y que construya la consulta de manera dinámica.
3. El resultado de la consulta se compara un formato de consulta válida y se rechaza en caso que no corresponda.

Este método tiene algunas desventajas, como reconoce el mismo autor, solo protege de inyecciones tautológicas y sobrecarga la base de datos con registros adicionales.

Al final de dicho año, se publicó otra revisión sistemática (Faker, Muslim, & Dachlan, 2017) sobre las técnicas usadas para realizar una inyección SQL. Este artículo contiene la clasificación más reciente de las inyecciones SQL y se encuentra resumida en la tabla 2.

Tabla 2 *Clasificación reciente de las inyecciones SQL*

Ref.	Tipos de clasificación	Técnicas / implementación	
[4] [6] [32] [3] [7] [2]	Ataques SQL Clásicos	Consulta a cuestas	Inserta consultas adicionales a ser ejecutadas
		Tautología	Crear una consulta que siempre será correcta
		Encoding alternativos	Ataques de codificación con objeto de evadir el filtrado. Ej. User=41444d494e en lugar de user=ADMIN
		Consultas lógicas / ilegales	Usar mensajes de error realizados por la base de datos para encontrar datos útiles
		Uniones	Consultas inyectadas son unidas con una consulta legal a través de la sentencia UNION
		Procedimientos almacenados	Ejecuta procedimientos internos o funciones
	Inferencial	Ataques SQL ciegos (Verdadero/ Falso)	Respuesta condicional
			Errores condicionales
			Canales fuera de banda
		Ataques SQL de tiempo (if,then)	Inyección doble ciega (retraso)
			Inyección profunda ciega (múltiples declaraciones)
	Ataques SQL específicos al motor	Huella digital de la base de datos (versión y host)	
		Mapeo de base de datos	
	Ataques SQL compuestos	Ataques SQL de fundición rápida	

Fuente: (Faker, Muslim & Dachlan, 2017)

En el año 2018, se realizaron investigaciones sobre las inyecciones de comandos y métodos de prevención y detección de las inyecciones SQL. En (Stasinopoulos & Ntantogian, 2018), los autores abordan de manera exhaustiva las inyecciones de comandos de sistema operativo. En este artículo, se presentan los diferentes tipos de

inyecciones de comando: Clásicas basadas en resultados, evaluación dinámica de código, ciegas: basadas en tiempo y en archivos permanentes y temporales. Luego presentan una herramienta para la evaluación de este riesgo en las aplicaciones web.

En una investigación realizada en la Universidad de San José, California (Ross, Moh, & Yao, 2018) se aplica las técnicas de Machine Learning y de análisis de datos multi-fuente, a la prevención de los ataques de inyecciones SQL. A través de estas técnicas, lograron detectar con un 98% de eficacia estos ataques.

Desde el año 2002 en que se documentó por primera vez el riesgo de inyecciones SQL inferenciales (Anley, 2002a) hasta el día de hoy se han realizado esfuerzos por reducir el tiempo en las pruebas de penetración y además, se han producido herramientas para automatizarlas. Sobre el riesgo de inyecciones inferenciales, son pocos los artículos escritos y los análisis realizados, aunque puede verse que el tema es de interés científico y profesional. Hasta la fecha, no se han realizado análisis teóricos sobre los algoritmos usados para el riesgo mencionado ni sobre las herramientas específicas utilizadas. Esta falta de estudios y la construcción de herramientas de manera independiente y sin rigor científico aumentan las posibilidades de reducir el tiempo de evaluación del riesgo de inyecciones SQL inferenciales.

3. Análisis y comparación de los algoritmos para la evaluación de inyecciones SQL inferenciales

Los ataques inferenciales se basan en razonamiento lógico y se infiere el resultado basado en el comportamiento del objetivo (W. G. J. Halfond et al., 2008; Štampar, 2016). Cualquier característica observable que presente cambios en el sitio web es utilizada para establecer un criterio de comportamiento cuando la inyección es falsa y otro cuando la inyección es verdadera. Pueden ser cambios en el código de retorno, en el contenido de la página (mostrando información diferente o errores de aplicación) o diferencias en el tiempo de respuesta. La característica principal de este tipo de ataque es que se introducen preguntas booleanas en la consulta original para que de acuerdo al resultado (verdadero o falso) realice el cambio en el comportamiento y de esa manera se pueda inferir el valor de la base de datos. Se han identificado dos subtipos para la realización de este ataque.

3.1 Tipos de inyecciones inferenciales

3.1.1 Basadas en tiempo

En este subtipo de ataque, la consulta original se modifica para que se introduzca un retraso de tiempo (Slaviero, 2012). Existen dos métodos establecidos para introducirlo:

- Funciones de pausa en la base de datos.
- Forzar la ejecución de consultas pesadas.

Sea cual sea el método elegido, a través de las entradas de la página web, se modifica la consulta SQL original que se realiza sobre la base de datos, de manera que se ejecuta una consulta alterada. Esta consulta contiene la pregunta booleana que permite extraer el contenido de la base de datos carácter por carácter. Si el resultado es verdadero, la base de datos ejecutará la pausa deseada, reflejándose en el tiempo de respuesta de la página web, indicando así el resultado de la pregunta booleana inyectada (Štampar, 2016). Con el fin de dar mayor claridad: si se tiene una página web conectada a una base de datos Postgres, el comando de SQL propietario PG_SLEEP, inyecta una cantidad de tiempo determinado. Entonces, se puede alterar la consulta original a través de las entradas de la aplicación así:

Consulta original:

*SELECT * FROM users WHERE login='admin' and password='admin'*

Consulta modificada (inyectando en el campo *password*):

*SELECT * FROM users WHERE login='' and password='' AND 1= (SELECT pg_sleep(5) WHERE substring(login,1,1)='a').*

En esta consulta modificada, la base de datos generará una pausa de 5 segundos en caso que el primer carácter del campo *login* sea la letra 'a'. A través de un mecanismo de medición del tiempo de carga de la página, se puede saber si el sitio web tuvo un tiempo de carga normal o uno con 5 segundos adicionales. En caso que no se refleje el tiempo de carga, se prosigue con la letra 'b' y así sucesivamente hasta que el tiempo de carga refleje la pausa de 5 segundos.

3.1.2 No basadas en tiempo

En esta tesis, se usa el término no basadas en tiempo, en lugar de basadas en error (Slaviero, 2012) o inyecciones ciegas (W. G. J. Halfond et al., 2008) o inyecciones ciegas no basadas en tiempo (Štampar, 2016). Aunque todos se refieren al mismo subtipo, la expresión basadas en error no incluyen los cambios en la página web diferentes a

mensajes de error (como cuándo se altera el código de retorno), el término inyección ciega, aunque es el más común, tiene una definición que también aplica para el subtipo basado en tiempo: “La información debe ser inferida desde el comportamiento de la página web al realizar preguntas de verdadero o falso al servidor” (W. G. J. Halfond et al., 2008, p. 5). Esta es, de hecho, una definición general a todas las inyecciones SQL inferenciales. Igual puede decirse de la definición de inyecciones ciegas no basadas en tiempo: “Los atacantes intentan vincular la parte de la pregunta inferencial a la declaración de SQL vulnerable de tal manera que cambia el resultado final dependiendo de la respuesta a la misma pregunta” (Štampar, 2016, p. 319).

En este subtipo, se modifica la consulta para forzar cualquier cambio en la página web que sea diferente al tiempo de ejecución. Un cambio común es introducir o forzar un error de ejecución sobre la consulta, como errores de conversión o de desbordamiento de tipo, división por cero o cualquier otro que solo se ejecutará si la consulta inyectada es verdadera. La consulta modificada quedaría así:

Consulta modificada (inyectando en el campo *password*):

*SELECT * FROM users WHERE login=’’ and password=’’ AND (SELECT 5/0 WHERE substring(login,1,1)=’a’).*

En esta consulta modificada, la base de datos generará un error de división por cero en caso que el primer carácter del campo *login* sea la letra ‘a’. A través de un mecanismo de análisis del código fuente de la página, se puede saber si el sitio web tuvo un cambio en su comportamiento, como un mensaje de error. En caso que no se refleje el error en la aplicación, se prosigue con la letra ‘b’ y así sucesivamente hasta que el código fuente interpretado refleje el error producido por la inyección.

3.2 Condicionales

Para la evaluación del este riesgo, debe elegirse la forma de usar o simular un condicional en la base de datos. Se identifican las siguientes tres formas de hacerlo.

3.2.1 Orden de evaluación sentencia AND

Las bases de datos deben elegir un orden para la evaluación de cada expresión del *WHERE* o de la sentencia *HAVING*. Suponiendo una consulta *WHERE* así: *expr1 AND expr2 AND expr3 AND expr4*. El motor de base de datos debe elegir un orden, iniciar desde *expr1* o desde *expr4* y evaluar cada expresión. Para el caso de las bases de datos que inician en el orden de aparición de las expresiones, esta inicia con *expr1* y si es verdadera, seguirá evaluando *expr2*. Si esta, a su vez, es verdadera, continúa con *expr3* y así sucesivamente. Existen motores que realizan esta evaluación en otro orden, como, por ejemplo, iniciar desde la expresión final y luego con la anterior y así sucesivamente hasta llegar a la primera.

Esta propiedad de la base de datos es aprovechada como condicional. Conociendo el orden del motor de base de datos de la página web vulnerable, se puede colocar la condición (pregunta booleana que extrae la información) que queremos evaluar antes o después (según sea el orden de evaluación) de la condición que genera el cambio en la página (Štampar, 2016).

3.2.2 Funciones condicionales de la base de datos

Algunas bases de datos tienen expresiones condicionales que permite ejecutar cierta instrucción si cumple alguna condición en particular. Por ejemplo, en el caso del motor PostgreSQL, existe la expresión *CASE*. En Oracle, se encuentra la misma expresión CASE y, además, la expresión *IF*. Esto último ocurre también en MYSQL y en SQL Server.

El siguiente ejemplo usa la función *CASE*, que es común la mayoría de motores de base de datos. Estas expresiones pueden usarse en el *SELECT*, en el *WHERE* o inclusive, en el *ORDER BY* (Štampar, 2016). Al tener la siguiente consulta original:

Consulta original:

*SELECT * FROM users WHERE login=’’ and password=’’*

Esta consulta es modificada para que, usando la expresión *CASE*, solo inyecte una pausa en la base de datos si el primer carácter del campo login es la letra A, así:

Consulta modificada

*SELECT * FROM users WHERE login=’’ and password=’’ AND (SELECT CASE WHEN SUBSTRING(login,1,1)=’A’ THEN pg_sleep(5) END) = ‘1’*

De esta manera, si la página demora cinco segundos más de lo normal en cargar, es porque el primer carácter del *login* es la letra ‘A’.

3.2.3 Ubicación en el WHERE de una SUB-CONSULTA

Además de las anteriores, que están documentadas, esta investigación define una manera adicional: Ubicación de la condición en la sección WHERE de una sub-consulta. Algunos motores de base de datos SQL permiten sub-consultas. Es posible realizar una sub-consulta que tenga la condición que queremos evaluar en la sección *WHERE* y la inyección escogida en la sección *SELECT*.

Para ilustrar este condicional, tomando la misma consulta original del condicional anterior, es posible modificarla para que se envíe a la base de datos así:

Consulta modificada

*SELECT * FROM USERS WHERE LOGIN=’’ and PASSWORD=’’ AND (SELECT PG_SLEEP(1) FROM users WHERE SUBSTRING(login,1,1)=’a’) = ‘1’*.

Sub-consulta (Para facilidad de lectura)

SELECT PG_SLEEP(1) FROM users WHERE SUBSTRING(login,1,1)=’a’

En este ejemplo, al evaluar la sub-consulta solo se ejecutaría la expresión de la sección *SELECT* si se cumple la condición de la sección *WHERE*. De esta manera puede usarse cómo un condicional para la inyección inferencial.

3.3 Algoritmos de extracción

Para la extracción de la información a través del riesgo de inyecciones SQL inferenciales, se han implementado los siguientes algoritmos.

3.3.1 Búsqueda Secuencial

Este algoritmo es el más intuitivo y sencillo de implementar. Consiste en realizar una búsqueda exhaustiva por cada carácter de la base de datos. A continuación, estará un ejemplo de cómo funciona dicho algoritmo.

Consulta original:

*SELECT * FROM users WHERE login='' and password=''*

Para extraer de la tabla *users* el campo *login* se inicia con la siguiente consulta modificada:

Consulta modificada 1:

*SELECT * FROM USERS WHERE LOGIN='' and PASSWORD='' AND (SELECT sleep(5) FROM users WHERE SUBSTRING(login,1,1)='a') = '1'*.

Si el tiempo de respuesta del sitio web es igual o superior a cinco segundos, entonces es porque el primer carácter de ese campo es la letra 'a'. En caso de que el tiempo sea inferior, se procede con la siguiente consulta modificada.

Consulta modificada 2:

*SELECT * FROM USERS WHERE LOGIN=’’ and PASSWORD=’’ AND (SELECT sleep(5) FROM users WHERE SUBSTRING(login,1,1)=’b’) = ‘1’*.

Y se repite el proceso, verificar el tiempo de respuesta, y así sucesivamente hasta que la página web responda con un tiempo de carga de más de cinco segundos. En ese caso, se procede con el siguiente carácter del campo. Así:

Consulta modificada 3:

*SELECT * FROM USERS WHERE LOGIN=’’ and PASSWORD=’’ AND (SELECT sleep(5) FROM users WHERE SUBSTRING(login,**2**,1)=’a’) = ‘1’*.

Y así sucesivamente por cada carácter del campo *login*, hasta encontrar la palabra completa.

En la tabla 3 se encuentra el pseudocódigo del algoritmo secuencial. En este algoritmo, la variable *n* indica la cantidad de caracteres a extraer.

Tabla 3 *Algoritmo secuencial*

```
1) InferentialSeqSearch(n):
        2) chars = getPossibleChars();
        3) value = “”
        4) for pos in range(1,n):
                5) for char in chars:
                        6) payload = getPayload(pos,char)
                        7) if(testInjection(payload)):
                                8) value += char
                                9) Break
10) return value
```

Fuente: Elaboración propia

En la línea 2, el método *getPossibleChars()*, retorna la lista de los posibles caracteres a comprobar campo por campo. Dependiendo del idioma de la base de datos y si usa caracteres especiales o no, este método retorna un subconjunto de los caracteres UNICODE, organizados en orden ascendente, descendente o cualquier otro orden de acuerdo a las optimizaciones implementadas. A continuación, por cada carácter a extraer,

se empieza a recorrer el sub-conjunto UNICODE. El método *getPayload(pos,char)* retorna la consulta SQL modificada para la extracción del carácter determinado. El método *testInjection(payload)* envía la inyección y analiza el comportamiento de la página web para determinar si es verdadero o falso. En caso de ser verdadero, se sigue con el siguiente carácter y se adiciona el encontrado a la cadena de retorno.

▪ Análisis Teórico

El análisis teórico que se muestra a continuación se basa en el algoritmo mostrado en la tabla 3. Las notaciones y variables usadas se explican en la lista de símbolos y abreviaturas al inicio de este documento.

En la tabla 4, se encuentra el tiempo de ejecución línea por línea del algoritmo secuencial. De acuerdo con esto, el tiempo de ejecución del algoritmo sería el siguiente:

$$T(n) = 1 + 1 + (n) + (n * \mathrm{A}) + (n * \mathrm{A}) + (n * \mathrm{A} * \delta) + (n) + (n) + 1$$

$$T(n) = 3 + 3(n) + 2(n\mathrm{A}) + (n\mathrm{A}\delta)$$

$$\boldsymbol{T(n) = n(\mathrm{A}[2 + \delta] + 3) + 3}$$

Tabla 4 *Tiempo de ejecución algoritmo secuencial*

Línea	Instrucción	Tiempo de ejecución
1	InferentialSeqSearch(n):	
2	chars = getPossibleChars();	1
3	value = “”	1
4	for pos in range(1,n):	n
5	for char in chars:	$n * \mathrm{A}$
6	payload = getPayload(pos,char)	$n * \mathrm{A}$
7	if(testInjection(payload)):	$n * \mathrm{A} * \delta$
8	value += char	n
9	break	n
10	return value	1

Fuente: Elaboración propia

Aunque el tiempo de ejecución calculado es cierto en el caso de las inyecciones inferenciales no basadas en tiempo, para el caso de las basadas en tiempo, es necesario adicionar el tiempo de inyección (Δ). Para dejar este punto claro, en la tabla 5 se encuentra el tiempo de ejecución del algoritmo secuencial basado en tiempo.

Adicionando el tiempo de la pausa inyectada, la ejecución total del algoritmo secuencial basado en tiempo sería el siguiente:

$$T(n) = 1 + 1 + \mathrm{n} + (n * \mathrm{A}) + (n * \mathrm{A}) + n(\mathrm{A} * \delta + \Delta) + \mathrm{n} + \mathrm{n} + 1$$
$$T(n) = 3 + 3(n) + 2(n\mathrm{A}) + (n\mathrm{A}\delta) + (n\Delta)$$
$$T(n) = 3 + n(3 + 2\mathrm{A} + \mathrm{A}\delta + \Delta)$$
$$\boldsymbol{T(n) = n(\mathrm{A}(2 + \delta) + 3 + \Delta) + 3}$$

Tabla 5 *Tiempo de ejecución algoritmo secuencial basado en tiempo*

Línea	Instrucción	Tiempo de ejecución
1	InferentialSeqSearch(n):	1
2	chars = getPossibleChars();	1
3	value = “”	1
4	for pos in range(1,n):	n
5	for char in chars:	$n * \mathrm{A}$
6	payload = getPayload(pos,char)	$n * \mathrm{A}$
7	if(testInjection(payload)):	$n * (\mathrm{A} * \delta + \Delta)$
8	value += char	n
9	break	n
10	return value	1

Fuente: Elaboración propia

3.3.2 Búsqueda Binaria

Este algoritmo, también llamado disección, se ha implementado en los algoritmos inferenciales y es una adaptación del algoritmo de búsqueda binaria sobre un arreglo

ordenado. Consiste en dividir repetidamente a la mitad los caracteres del sub-conjunto de caracteres que podrían contener el carácter a encontrar. A continuación, se esboza un ejemplo de su funcionamiento.

Consulta original:

*SELECT * FROM users WHERE login='' and password=''*

Para extraer de la tabla *users* el campo *login* se inicia con la siguiente consulta modificada:

Consulta modificada 1:

*SELECT * FROM USERS WHERE LOGIN='' and PASSWORD='' AND (SELECT sleep(5) FROM users WHERE ASCII(SUBSTRING(login,1,1))>128) = '1'*.

Si el tiempo de respuesta del sitio web es igual o superior a cinco segundos, entonces es porque el equivalente UNICODE del primer carácter de ese campo es superior a 128. En caso de que el tiempo sea inferior, quiere decir que el carácter se encuentra entre el valor UNICODE 1 y 128, por lo tanto, se procede con la siguiente consulta modificada.

Consulta modificada 2:

*SELECT * FROM USERS WHERE LOGIN='' and PASSWORD='' AND (SELECT sleep(5) FROM users WHERE ASCII(SUBSTRING(login,1,1))>64) = '1'*.

En caso de que el tiempo de respuesta sea igual o mayor a los cinco segundos inyectados, el valor equivalente UNICODE del primer carácter de ese campo está entre 65 y 128. En caso contrario, el valor estaría entre 1 y 64. Se siguen realizando consultas similares, dividiendo entre dos las posibilidades hasta encontrar el primer carácter del campo. Luego se continúa con el siguiente carácter hasta encontrar todos los elementos de la base de datos.

En la tabla 6 se encuentra la implementación de este algoritmo con una notación similar a la mostrada en el caso de la búsqueda secuencial. Al igual que en el primero algoritmo, en

la línea 2, el método *getPossibleChars()*, retorna la lista de los posibles caracteres a comprobar campo por campo. A continuación, por cada carácter a extraer, se define la cota inferior y superior (líneas 5 y 6).

Tabla 6 *Algoritmo de búsqueda binaria*

```
1) BinSearch(n):
        2) chars = getPossibleChars();
        3) value = ""
        4) for pos in range(1,n):
                5) L = 0
                6) R = len(chars)-1
                7) while L < R:
                        8) T=chars[L+math.floor((R-L)/2)]
                        9) payload=getPayload(pos,T,'=')
                        10) If(testInjection(payload))
                                11) value += T
                                12) Break
                        13) payload=getPayload(pos,char,'>')
                        14) If testInjection(payload):
                                15) L=L+math.floor((R-L)/2)+1
                        16) else
                                17) R=L+math.floor((R-L)/2)-1
18) return value
```

Fuente: Elaboración propia

Las cotas iniciales serán desde 0 hasta la cantidad máxima de caracteres retornados en la línea 2. Posteriormente, mientras la cota inferior no sea mayor a la superior, se obtiene el elemento de la mitad del arreglo (línea 8). En la línea 9 se genera una consulta modificada para verificar si este carácter de la mitad es el elemento buscado. Esta consulta modificada se envía al servidor y se verifica a través del comportamiento esperado si se encontró el carácter (línea 10). En caso de ser cierto, se adiciona el carácter encontrado a la variable de retorno y se continúa con el siguiente carácter. En caso de no ser cierto, se genera una consulta modificada que pregunte si el carácter a encontrar es mayor al

carácter de la mitad del sub-arreglo, calculado en la línea 8. Se envía esta consulta al servidor y en caso de ser cierta se modifica la cota inferior para hacerlo igual al carácter de la mitad. En caso de no ser cierto, se modifica la cota superior para hacerlo igual a este mismo carácter. Finalmente, se retorna la variable con la cadena encontrada.

▪ Análisis teórico

El análisis teórico que se muestra a continuación se basa en el algoritmo mostrado en la tabla 6. Al igual que en el caso anterior, y en todos los casos posteriores, las notaciones y variables usadas se explican en la lista de símbolos y abreviaturas.

En la tabla 7 se encuentra línea por línea el tiempo de ejecución del algoritmo de búsqueda binaria. Por lo tanto, el tiempo de ejecución de este algoritmo es el siguiente:

$$T(n) = 1 + 1 + (n) + (n) + (n) + \left(n * \frac{\mathrm{A}}{2^i}\right) + \left(n * \frac{\mathrm{A}}{2^i}\right) + \left(n * \frac{\mathrm{A}}{2^i}\right) + \left(n * \frac{\mathrm{A}}{2^i} * \delta\right) + (n) + (n) + \left(n * \frac{\mathrm{A}}{2^i}\right) + \left(n * \frac{\mathrm{A}}{2^i} * \delta\right) + \frac{\left(n * \frac{\mathrm{A}}{2^i}\right)}{2} + \left(n * \frac{\mathrm{A}}{2^i}\right) + \frac{\left(n * \frac{\mathrm{A}}{2^i}\right)}{2} + 1$$

$$T(n) = 3 + 5n + \left(6n\frac{\mathrm{A}}{2^i}\right) + \left(2n * \frac{\mathrm{A}}{2^i} * \delta\right)$$

$$T(n) = n\left(5 + 6\left(\frac{\mathrm{A}}{2^i}\right) + 2\left(\frac{\mathrm{A}}{2^i}\delta\right)\right) + 3$$

$$T(n) = n\left(\frac{\mathrm{A}}{2^i}(2\delta + 6) + 5\right) + 3$$

Tabla 7 *Tiempo de ejecución del algoritmo de búsqueda binaria*

Línea	Instrucción	Tiempo de ejecución
1	BinSearch(n):	
2	chars = getPossibleChars();	1
3	value = “”	1
4	for pos in range(1,n):	n
5	L = 0	n
6	R = len(chars)-1	n
7	while L _ R	$\left(n * \frac{\mathrm{A}}{2^i}\right)$ i = número de iteraciones para encontrar el elemento
8	T=chars[L+math.floor((R-L)/2)]	$\left(n * \frac{\mathrm{A}}{2^i}\right)$
9	payload=getPayload(pos,T,’=’)	$\left(n * \frac{\mathrm{A}}{2^i}\right)$
10	If(testInjection(payload))	$\left(n * \frac{\mathrm{A}}{2^i} * \delta\right)$
11	value += T	(n)
12	Break	(n)
13	payload=getPayload(pos,char,’>’)	$\left(n * \frac{\mathrm{A}}{2^i}\right)$
14	If testInjection(payload):	$\left(n * \frac{\mathrm{A}}{2^i} * \delta\right)$
15	L=L+math.floor((R-L)/2)+1	$\frac{\left(n * \frac{\mathrm{A}}{2^i}\right)}{2}$
16	else	$\left(n * \frac{\mathrm{A}}{2^i}\right)$
17	R=L+math.floor((R-L)/2)-1	$\frac{\left(n * \frac{\mathrm{A}}{2^i}\right)}{2}$
18	return value	1

Fuente: Elaboración propia

En el peor de los casos: i = m (el número máximo de iteraciones)

El elemento es encontrado en la última división binaria, donde el arreglo resultante es de tamaño 1. Entonces:

$$\frac{\mathrm{A}}{2^m} <= 1$$

$$\mathrm{A} <= 2^m$$

$$\log_2 \mathrm{A} > m$$

Por lo tanto, la cantidad de iteraciones en la línea 7 es menor o igual a $\log_2 \mathrm{A}$

De acuerdo con estos cálculos, el tiempo de ejecución es:

$$\boldsymbol{T(n) = n(\log_2 \mathrm{A}\,(2\delta + 6) + 5) + 3}$$

Al igual que sucede con el algoritmo secuencial, es necesario analizar el caso de la inyección de un retraso de tiempo para evaluar el impacto de dicho tiempo en la ejecución del algoritmo. El cambio sobre el tiempo de ejecución se ve reflejado en la tabla 8. De acuerdo con esta tabla, el tiempo de ejecución es el siguiente:

$$T(n) = 1 + 1 + (n) + (n) + (n) + \left(n * \frac{\mathrm{A}}{2^i}\right) + \left(n * \frac{\mathrm{A}}{2^i}\right) + \left(n * \frac{\mathrm{A}}{2^i}\right) + n\left(\frac{\mathrm{A}}{2^i}\delta + \Delta\right) + (n) + (n) + \left(n * \frac{\mathrm{A}}{2^i}\right) + (n * \frac{\mathrm{A}}{2^i})(\delta + \Delta) + \left(\frac{\left(n * \frac{\mathrm{A}}{2^i}\right)}{2}\right) + \left(n * \frac{\mathrm{A}}{2^i}\right) + \left(\frac{\left(n * \frac{\mathrm{A}}{2^i}\right)}{2}\right) + 1$$

$$T(n) = 3 + 5(n) + 6\left(n\frac{\mathrm{A}}{2^i}\right) + n\left(\frac{\mathrm{A}}{2^i}\,\delta + \Delta\right) + (n\frac{\mathrm{A}}{2^i})(\delta + \Delta)$$

$$T(n) = 3 + 5n + 6\left(n\frac{\mathrm{A}}{2^i}\right) + \left(n\frac{\mathrm{A}}{2^i}\delta\right) + (\mathrm{n}\Delta) + \left(n\frac{\mathrm{A}}{2^i}\delta\right) + \left(n\frac{\mathrm{A}}{2^i}\Delta\right)$$

$$T(n) = 3 + n\left(5 + 6\left(\frac{\mathrm{A}}{2^i}\right) + 2\left(\frac{\mathrm{A}}{2^i}\delta\right) + \left(\frac{\mathrm{A}}{2^i}\Delta\right) + \Delta\right)$$

$$T(n) = n\left(\frac{\mathrm{A}}{2^i}(6 + 2\delta + \Delta) + \Delta + 5\right) + 3$$

Tabla 8 *Tiempo de ejecución del algoritmo de búsqueda binaria basado en tiempo*

Línea	Instrucción	Tiempo de ejecución
1	BinSearch(n):	
2	chars = getPossibleChars();	1
3	value = “”	1
4	for pos in range(1,n):	n
5	L = 0	n
6	R = len(chars)-1	n
7	while L < R	$\left(n * \frac{\mathrm{A}}{2^i}\right)$ i = número de iteraciones para encontrar elemento
8	T=chars[L+math.floor((R-L)/2)]	$\left(n * \frac{\mathrm{A}}{2^i}\right)$
9	payload=getPayload(pos,T,’=’)	$\left(n * \frac{\mathrm{A}}{2^i}\right)$
10	If(testInjection(payload))	$\left(n * (\frac{\mathrm{A}}{2^i}\delta + \Delta)\right)$
11	value += T	(n)
12	Break	(n)
13	payload=getPayload(pos,char,’>’)	$\left(n * \frac{\mathrm{A}}{2^i}\right)$
14	If testInjection(payload):	$\left(n * \frac{\mathrm{A}}{2^i} * (\delta + \Delta)\right)$
15	L=L+math.floor((R-L)/2)+1	$\frac{\left(n * \frac{\mathrm{A}}{2^i}\right)}{2}$
16	else	$\left(n * \frac{\mathrm{A}}{2^i}\right)$
17	R=L+math.floor((R-L)/2)-1	$\frac{\left(n * \frac{\mathrm{A}}{2^i}\right)}{2}$
18	return value	1

Fuente: Elaboración propia

En el peor de los casos: i = m (el número máximo de iteraciones)

El elemento es encontrado en la última división binaria, donde el arreglo resultante es de tamaño igual a 1. Entonces:

$$\frac{\mathrm{A}}{2^m} <= 1$$

$$\mathrm{A} <= 2^m$$

$$\log_2 \mathrm{A} > m,$$

Por lo tanto, la cantidad de iteraciones en la línea 7 es menor o igual al $\log_2 \mathrm{A}$

De acuerdo con estos cálculos, el tiempo de ejecución es:

$$\boldsymbol{T(n) = n(\log_2 \mathrm{A}\,(2\delta + 6 + \Delta) + \Delta + 5) + 3}$$

3.3.3 Extracción bit a bit

Este algoritmo consiste en tomar el octeto de bits que conforman un carácter en la base de datos y extraerlo bit a bit a través de preguntas inyectadas a la consulta. A continuación, se explica su funcionamiento.

Consulta original:

*SELECT * FROM users WHERE login='' and password=''*

Para extraer de la tabla *users* el campo *login* se inicia con la siguiente consulta modificada:

Consulta modificada 1:

*SELECT * FROM USERS WHERE LOGIN='' and PASSWORD='' AND (select sleep(5) WHERE (1=ASCII(SUBSTRING((login),1,1))&128)) = '1'*.

Esta consulta modificada, realiza una función de bits llamada “AND”. La función AND solo retorna 1 cuándo ambos elementos operados son 1. En el siguiente ejemplo se muestra su uso en la extracción de bits:

Suponiendo que el carácter a extraer es la letra ‘c’:

Equivalente UNICODE de la letra c es 99. El número 99 en su equivalente en bits es 01100011. En el ejemplo de la consulta modificada 1, se realizar la operación AND con el número 128, cuyo equivalente en bits es 10000000. Por lo tanto, al realizar esta operación, se tiene el siguiente resultado:

01100011 AND 10000000 = 00000000 = 0.

Por lo tanto, si el tiempo de respuesta del sitio web es igual o superior a cinco segundos, entonces es porque el bit más significativo del equivalente UNICODE del primer carácter del campo a extraer es 1. En caso de que el tiempo sea inferior, quiere decir que dicho bit es 0. Después de esto, se continúa con el siguiente bit.

Consulta modificada 2:

*SELECT * FROM USERS WHERE LOGIN=’’ and PASSWORD=’’ AND (select sleep(5) WHERE (1=ASCII(SUBSTRING((login),1,1))&64)) = ‘1’*.

Cuando se realiza está pregunta con el número 64, en bits 01000000, se extrae el siguiente bit. Y así sucesivamente, se realiza el mismo proceso con 32 (00100000), 16 (00010000), 8 (00001000), 4 (00000100), 2 (00000010) y 1 (00000001).

En la tabla 9 se muestra una implementación de dicho algoritmo. En la línea 2, se tiene el arreglo que permite la extracción de cada bit del carácter. Entonces, por cada carácter y por cada bit del mismo, se genera una cadena modificada (línea 7) donde contiene la pregunta inyectada por el bit a extraer de turno. En la línea 8 se envía la inyección y se analiza el comportamiento del sitio web para inferir si es verdadero o falso. En caso que sea verdadero, el bit es uno y en caso contrario, cero. Finalmente, la función bitsToChar (línea 12) retorna el equivalente UNICODE los bits extraídos.

Tabla 9 *Algoritmo bit a bit*

```
1) bitByBitSearch(n):
        2) 2pow = [128,64,32,16,8,4,2,1]
        3) value = ""
        4) for pos in range(1,n):
                5) bits = ""
                6) for i = 0 to sizeof(2pow):
                        7) payload=getPayload(pos, 2pow[i])
                        8) If(testInjection(payload))
                                9) bits += "1"
                        10) else
                                11) bits += "0"
                12) value+=bitsToChar(bits)
                13) return value
```

Fuente: Elaboración propia

▪ Análisis teórico

El análisis teórico que se muestra a continuación se basa en el algoritmo mostrado en la tabla 9. En la tabla 10 se encuentra línea por línea el tiempo de ejecución del algoritmo de búsqueda binaria. Por lo tanto, el tiempo de ejecución de este algoritmo es el siguiente:

$$T(n) = 1 + 1 + n + n + (\log_2 \mathrm{A} * n) + (\log_2 \mathrm{A} * n) + (\log_2 \mathrm{A} * n * \delta) + \left(\frac{\log_2 \mathrm{A} * n}{2}\right) + (\log_2 \mathrm{A} * n) + \left(\frac{\log_2 \mathrm{A} * n}{2}\right) + n + 1$$

$$T(n) = 3 + 3n + 3(n \log_2 \mathrm{A}) + (n\delta \log_2 \mathrm{A}) + 2\left(\frac{n \log_2 \mathrm{A}}{2}\right)$$

$$T(n) = 3 + 3n + 3n \log_2 \mathrm{A} + (n\delta) \log_2 \mathrm{A} + n \log_2 \mathrm{A}$$

$$T(n) = 3 + 3n + 4n \log_2 \mathrm{A} + (n\delta) \log_2 \mathrm{A}$$

$$\boldsymbol{T(n) = \log_2 \mathrm{A}\,(n\delta + 4n) + 3n + 3}$$

Tabla 10 *Tiempo de ejecución algoritmo bit a bit*

Línea	Instrucción	Tiempo de ejecución
1	bitByBitSearch(n):	
2	2pow = [128,64,32,16,8,4,2,1]	1
3	value = “”	1
4	for pos in range(1,n):	n
5	bits = ""	n
6	for i = 0 to sizeof(2pow):	$\log_2 \mathrm{A} * n$
7	payload=getPayload(pos, 2pow[i])	$\log_2 \mathrm{A} * n$
8	If(testInjection(payload))	$\log_2 \mathrm{A} * n * \delta$
9	bits += “1”	$\log_2 \mathrm{A} * n/2$
10	else	$\log_2 \mathrm{A} * n$
11	bits += “0”	$\log_2 \mathrm{A} * n/2$
12	value+=bitsToChar(bits)	n
13	return value	1

Fuente: Elaboración propia

Al igual que los dos algoritmos anteriores, el tiempo de inyección sobre la pregunta modificada incide en el tiempo de ejecución del algoritmo para el caso de las inyecciones basadas en tiempo.

En la tabla 11, se encuentra el tiempo de ejecución incluyendo el tiempo inyectado.

De acuerdo con esta tabla, el tiempo de ejecución para el algoritmo bit a bit basado en tiempo es el siguiente:

$$T(n) = 1 + 1 + n + n + (\log_2 \mathrm{A} * n) + (\log_2 \mathrm{A} * n) + (\log_2 \mathrm{A} * n * (\delta + \Delta)) + \left(\frac{\log_2 \mathrm{A} * n}{2}\right) + (\log_2 \mathrm{A} * n) + \left(\frac{\log_2 \mathrm{A} * n}{2}\right) + n + 1$$

$$T(n) = 3 + 3n + 3(n \log_2 \mathrm{A}) + (n\delta) \log_2 \mathrm{A} + (n\Delta) \log_2 \mathrm{A} + 2\left(\frac{n \log_2 \mathrm{A}}{2}\right)$$

$$T(n) = 3 + 3n + 3n \log_2 \mathrm{A} + (n\delta) \log_2 \mathrm{A} + (n\Delta) \log_2 \mathrm{A} + n \log_2 \mathrm{A}$$

$$T(n) = 3 + 3n + 4n \log_2 \mathrm{A} + (n\delta) \log_2 \mathrm{A} + (n\Delta) \log_2 \mathrm{A}$$

$$\boldsymbol{T(n) = \log_2 \mathrm{A}\,(4\mathrm{n} + n\delta + n\Delta) + 3\mathrm{n} + 3}$$

Tabla 11 *Tiempo de ejecución del algoritmo bit a bit basado en tiempo*

Línea	Instrucción	Tiempo de ejecución
1	bitByBitSearch(n):	
2	2pow = [128,64,32,16,8,4,2,1]	1
3	value = “”	1
4	for pos in range(1,n):	n
5	bits = ""	n
6	for i = 0 to sizeof(2pow):	$\log_2 \mathrm{A} * n$
7	payload=getPayload(pos, 2pow[i])	$\log_2 \mathrm{A} * n$
8	If(testInjection(payload))	$\log_2 \mathrm{A} * n * (\delta + \Delta)$
9	bits += “1”	$\log_2 \mathrm{A} * n/2$
10	else	$\log_2 \mathrm{A} * n$
11	bits += “0”	$\log_2 \mathrm{A} * n/2$
12	value+=bitsToChar(bits)	n
13	return value	1

Fuente: Elaboración propia

3.4 Comparación de los algoritmos de extracción

En la sección 3.3 se explicaron los algoritmos de extracción y se calcularon los tiempos de ejecución de cada algoritmo basados en la metodología definida. Usando esos tiempos, es posible comparar cuál de los algoritmos de extracción es el que extrae los caracteres de la base de datos más eficiente de manera teórica.

Se realizará una comparación genérica, aunque, en algunos casos y para explicar de manera clara los resultados alcanzados, el valor de A será igual a 256. Esto significa que los caracteres de la base de datos pueden ser los 128 caracteres ASCII más los caracteres

especiales más utilizados, hasta el carácter UNICODE 256. En la tabla 12, se encuentra un resumen de los tiempos de ejecución de cada algoritmo para facilitar su lectura y la comprensión de las comparaciones realizadas.

Tabla 12 *Tiempo de inyección de los algoritmos de extracción*

Algoritmo	No basado en tiempo	Basados en tiempo
Secuencial	$T(n) = n(\mathrm{A}[2+\delta]+3)+3$	$T(n) = n(\mathrm{A}(2+\delta)+3+\Delta)+3$
Búsqueda binaria	$T(n) = n(\log_2 \mathrm{A}\,(2\delta+6)+5)+3$	$T(n) = n(\log_2 \mathrm{A}\,(2\delta+6+\Delta)+\Delta+5)+3$
Bit a bit	$T(n) = \log_2 \mathrm{A}\,(n\delta+4n)+3n+3$	$T(n) = \log_2 \mathrm{A}\,(4\mathrm{n}+n\delta+n\Delta)+3\mathrm{n}+3$

Fuente: Elaboración propia

3.4.1 Comparación de los algoritmos secuencial y binario

De acuerdo con la metodología definida, para cada comparación se realiza una hipótesis y posterior a ella, se realiza la comprobación o la negación de la misma de manera matemática y empírica. La comprobación empírica se realizará en la sección 3.4.4.

- **Hipótesis**

El tiempo de ejecución del algoritmo binario siempre es menor que el tiempo empleado por el algoritmo secuencial para las inyecciones inferenciales basadas en tiempo y las no basadas en tiempo.

- **Comprobación Matemática para las inyecciones no basadas en tiempo**

De acuerdo con el enunciado de la hipótesis:

$T(n)binario > T(n)secuencial$

$n(\mathrm{A}[2+\delta]+3)+3 > n(\log_2 \mathrm{A}\,(2\delta+6)+5)+3$

$\mathrm{A}[2+\delta]+3 > \log_2 \mathrm{A}\,(2\delta+6)+5$

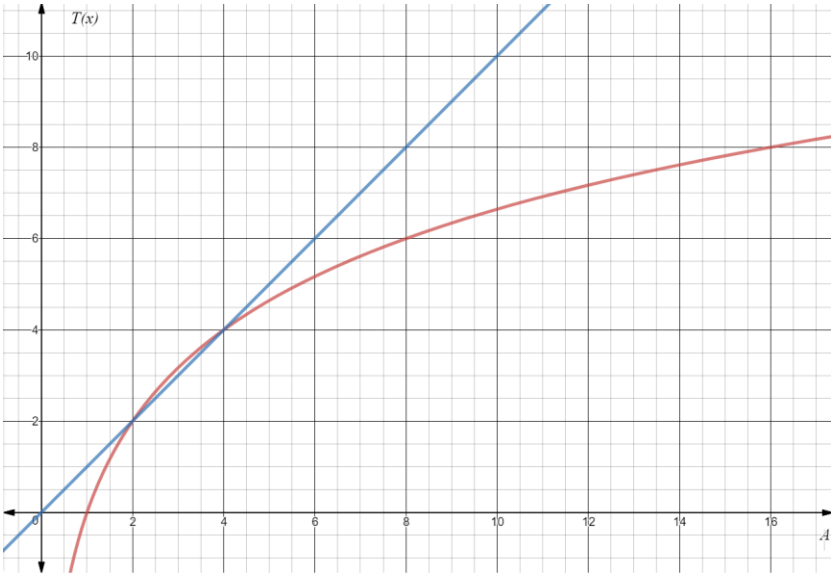
$2\mathrm{A}+\mathrm{A}\delta > 2\delta \log_2 \mathrm{A}+6\log_2 \mathrm{A}+2$

$\mathrm{A}\delta \geq 2\delta \log_2 \mathrm{A}$ y $2\mathrm{A} > 6\log_2 \mathrm{A}+2$

$\mathrm{A} \geq 2\delta \log_2 \mathrm{A}\ (1)$ y $\mathrm{A} > 3\log_2 \mathrm{A}+1\ (2)$

En la figura 3 se encuentra la gráfica de la primera ecuación. De acuerdo con esta, para que la desigualdad se cumpla, los valores de A deben ser mayores que cuatro ($\mathrm{A} > 4$). En esta figura, el eje X corresponde a la variable A y el eje Y al tiempo de ejecución. La ecuación graficada en color azul es $f(x) = A$ y la graficada en color rojo es $f(x) = 2\delta\, log_2\, A$. Después de A=4, el tiempo de la segunda ecuación nunca es mayor que la primera.

Figura 3 *Ecuación* $A \geq 2\delta\, log_2\, A$

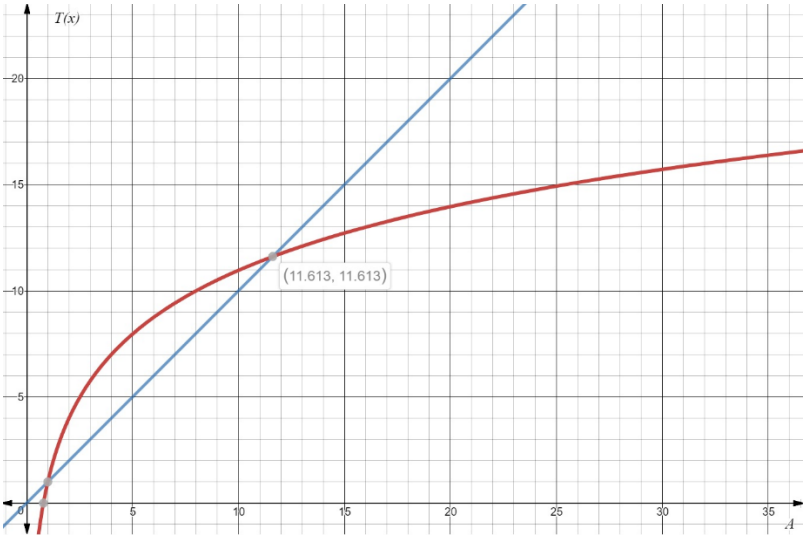


Fuente: Elaboración propia

La siguiente ecuación, número 2, se encuentra graficada en la figura 4. De acuerdo con esta, para que la desigualdad sea cierta, los valores de A deben ser mayores que 12. En esta figura, el eje X corresponde a la variable A y el eje Y al tiempo de ejecución. La

ecuación graficada en color azul es $f(x) = A$ y la graficada en color rojo es $f(x) = 3\,log_2\,A + 1$. Después de A=11.613 ~12, el tiempo de la segunda ecuación nunca es mayor que la primera.

Figura 4 *Ecuación* $A > 3\,log_2\,A + 1$



Fuente: Elaboración propia

- ## Comprobación Matemática para las inyecciones basadas en tiempo

Debido al tiempo de inyección, los tiempos de respuesta para ambos algoritmos son incrementados. Por tanto, la hipótesis debe probarse igualmente para el caso de las inyecciones basadas en tiempo.

De acuerdo con la hipótesis:

$$T(n)binario > T(n)secuencial$$

$$n(\mathrm{A}(2+\delta)+3+\Delta)+3 > n(\log_2 \mathrm{A}\,(2\delta+6+\Delta)+\Delta+5)+3$$

$$\mathrm{A}(2+\delta)+3+\Delta > \log_2 \mathrm{A}\,(2\delta+6+\Delta)+\Delta+5$$

$$\mathrm{A}(2+\delta) > \log_2 \mathrm{A}\,(2\delta+6+\Delta)+2$$

$$2\mathrm{A}+\mathrm{A}\delta > 2\delta\log_2 \mathrm{A}+6\log_2 \mathrm{A}+\Delta\log_2 \mathrm{A}+2$$

$A\delta > 2\delta \log_2 A + \Delta \log_2 A$ y $2A > 6 \log_2 A + 2$

$(A\delta - 2\delta \log_2 A)/\log_2 A > \Delta$ y $A > 3 \log_2 A + 1$

$(A\delta/\log_2 A - 2\delta) > \Delta$ y $A > 3 \log_2 A + 1$

$\Delta \leq \delta * (\frac{A}{\log_2 A} - 2)$ y $A \geq 12$

Para el caso de $A = 256$ Caracteres UNICODE:

$\Delta \leq \delta * (\frac{256}{\log_2 256} - 2)$

$\Delta \leq 30\delta$

- **Conclusión**

El tiempo de ejecución del algoritmo binario siempre es menor que el tiempo empleado por el algoritmo secuencial para las inyecciones inferenciales basadas en tiempo y las no basadas en tiempo siempre y cuando se cumpla que:

1. La cantidad de caracteres UNICODE a buscar deber ser mayor o igual a 12.
2. Para la extracción basada en tiempo, este tiempo inyectado debe ser menor al tiempo de respuesta del sitio web multiplicado por la diferencia entre la cantidad de caracteres UNICODE sobre el logaritmo en base dos de esta misma cantidad y dos: $\Delta \leq \delta * (\frac{A}{\log_2 A} - 2)$.

3.4.2 Comparación de los algoritmos secuencial y bit a bit

- **Hipótesis**

El tiempo de ejecución del algoritmo bit a bit siempre es menor que el tiempo empleado por el algoritmo secuencial para las inyecciones inferenciales basadas en tiempo y las no basadas en tiempo.

▪ Comprobación Matemática para las inyecciones no basadas en tiempo

De acuerdo con el enunciado de la hipótesis:

$$T(n)binario > T(n)secuencial$$

$$n(\mathrm{A}[2+\delta]+3)+3 > \log_2 \mathrm{A}\,(n\delta+4n)+3n+3$$

$$2n\mathrm{A}+\mathrm{nA}\delta+3\mathrm{n} > \log_2 \mathrm{A}\,(n\delta+4n)+3n$$

$$2n\mathrm{A}+\mathrm{nA}\delta > \log_2 \mathrm{A}\,(n\delta+4n)$$

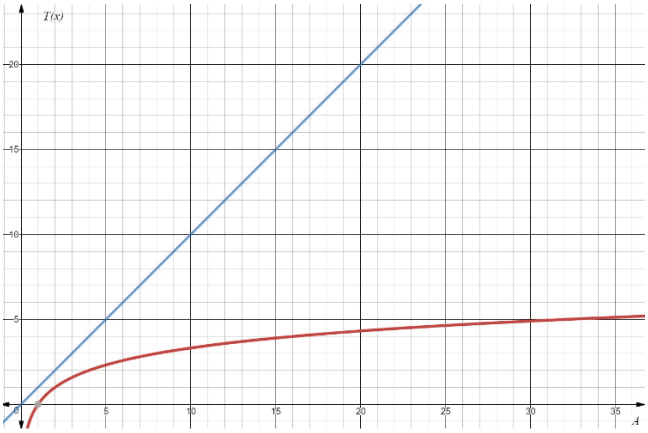
$$2n\mathrm{A}+\mathrm{nA}\delta > n\delta\log_2 \mathrm{A}+4n\log_2 \mathrm{A}$$

$$\mathrm{nA}\delta \geq n\delta\log_2 \mathrm{A} \text{ y } 2n\mathrm{A} > 4n\log_2 \mathrm{A}$$

$$\mathrm{A} \geq \log_2 \mathrm{A}\,(3) \text{ y } \mathrm{A} > 2\log_2 \mathrm{A}\ (4)$$

En la figura 5 se encuentra la gráfica de la tercera ecuación de estas comprobaciones.

Figura 5 *Ecuación* $A \geq log_2 A$



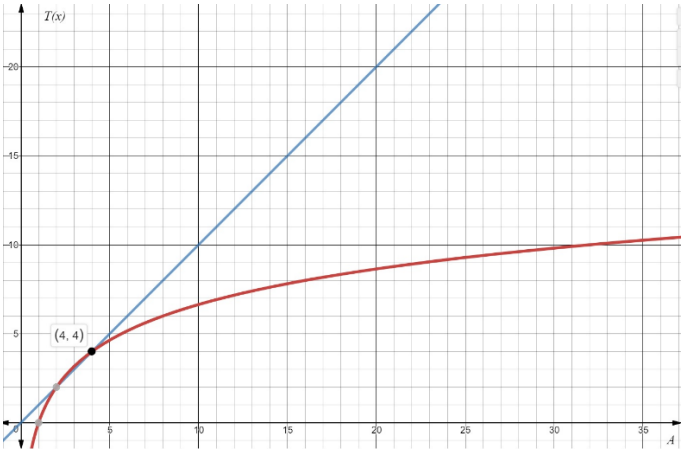
Fuente: Elaboración propia

De acuerdo con esta, la ecuación en cuestión siempre será cierta. En esta figura, el eje X corresponde a la variable A y el eje Y al tiempo de ejecución. La ecuación graficada en color azul es $f(x) = A$ y la graficada en color rojo es $f(x) = log_2 A$. Se puede notar que, para cada valor de A, la segunda ecuación nunca es mayor que la primera.

La siguiente ecuación, número 4, se encuentra graficada en la figura 6. De acuerdo con esta, para que la desigualdad sea cierta, los valores de A deben ser mayores que 4. En

esta figura, la ecuación graficada en color azul es $f(x) = A$ y la graficada en color rojo es $f(x) = 2log_2 A$. Se puede notar que, después de A=4, la segunda ecuación nunca es mayor que la primera.

Figura 6 *Ecuación* $A > 2\,log_2 A$



Fuente: Elaboración propia

Por lo tanto, para el caso de las inferenciales no basados en tiempo, el tiempo de respuesta del algoritmo bit a bit siempre será mejor que el tiempo del algoritmo secuencial para una cantidad de valores UNICODE superiores a 4.

- ## Comprobación Matemática para las inyecciones basadas en tiempo

De acuerdo con el enunciado de la hipótesis:

$$T(n)binario > T(n)secuencial$$

$$n(\mathrm{A}(2+\delta)+3+\Delta)+3 > \log_2 \mathrm{A}\,(4\mathrm{n}+n\delta+n\Delta)+3\mathrm{n}+3$$

$$2\mathrm{A}n+n\mathrm{A}\delta+3n+n\Delta > \log_2 \mathrm{A}\,(4\mathrm{n}+n\delta+n\Delta)+3\mathrm{n}$$

$$2\mathrm{A}n+n\mathrm{A}\delta+n\Delta > 4\mathrm{n}\log_2 \mathrm{A}+n\delta\log_2 \mathrm{A}+n\Delta\log_2 \mathrm{A}$$

$$2\mathrm{A}+\mathrm{A}\delta+\Delta > 4\log_2 \mathrm{A}+\delta\log_2 \mathrm{A}+\Delta\log_2 \mathrm{A}$$

$$2\mathrm{A}+\mathrm{A}\delta-4\log_2 \mathrm{A}-\delta\log_2 \mathrm{A} > \Delta\log_2 \mathrm{A}-\Delta$$

$$\Delta(\log_2 A - 1) \leq 2A + A\delta - 4\log_2 A - \delta \log_2 A$$

$$\Delta \leq \frac{2A + A\delta - 4\log_2 A - \delta \log_2 A}{\log_2 A - 1}$$

$$\Delta \leq \frac{A(\delta + 2) - \log_2 A\,(\delta + 4)}{\log_2 A - 1}$$

Para el caso de $A = 256$ caracteres UNICODE:

$$\Delta \leq \frac{A(\delta + 2) - \log_2 A\,(\delta + 4)}{\log_2 A - 1}$$

$$\Delta \leq \frac{256(\delta + 2) - 8(\delta + 4)}{8 - 1}$$

$$\Delta \leq \frac{256\delta + 512 - 8\delta - 32}{7}$$

$$\Delta \leq \frac{248\delta + 480}{7}$$

$$\Delta \leq \frac{248\delta}{7} + \frac{480}{7}$$

$$\Delta \leq 35{,}42857\delta + 68{,}5714$$

Por esto, en el caso de las inyecciones inferenciales basadas en tiempo, la hipótesis está supeditada al tiempo de cargue de la página web δ.

▪ Conclusión

El tiempo de ejecución del algoritmo bit a bit es menor que el tiempo empleado por el algoritmo secuencial para las inyecciones inferenciales no basadas en tiempo si se cumple la siguiente condición:

La cantidad de caracteres UNICODE a buscar sea superior a 4 ($A > 4$)

El tiempo de ejecución del algoritmo bit a bit es menor que el tiempo empleado por el algoritmo secuencial para las inyecciones inferenciales basadas en tiempo si se cumple la siguiente condición:

El tiempo inyectado debe ser menor a la siguiente expresión $\Delta \leq \frac{A(\delta+2)-\log_2 A(\delta+4)}{\log_2 A-1}$.

3.4.3 Comparación de los algoritmos bit a bit y binario

- **Hipótesis**

El tiempo de ejecución del algoritmo bit a bit siempre es menor que el tiempo empleado por el algoritmo binario para las inyecciones inferenciales basadas en tiempo y las no basadas en tiempo.

- **Comprobación Matemática para las inyecciones no basadas en tiempo**

De acuerdo con el enunciado de la hipótesis:

$T(n)binario > T(n)secuencial$

$n(\log_2 A\,(2\delta+6)+5)+3 > n\log_2 A\,(\delta+4)+3n+3$

$n(\log_2 A\,(2\delta+6)+5)+3 > n(\log_2 A\,(\delta+4)+3)+3$

$\log_2 A\,(2\delta+6)+5 > \log_2 A\,(\delta+4)+3$

$\delta\log_2 A+6\log_2 A+2 > 4\log_2 A$

$\delta > \frac{-2\log_2 A-2}{\log_2 A}$

$\delta > -2-\frac{2}{\log_2 A}$

Por lo tanto, para el caso de las inferenciales no basados en tiempo, esto es real para tiempos de cargue de página superiores a $\delta > -2 - \frac{2}{\log_2 \mathrm{A}}$ unidades de procesamiento lo cual siempre será cierto debido a que $\delta > 0\ s$.

- **Comprobación Matemática para las inyecciones basadas en tiempo**

De acuerdo con la hipótesis:

$T(n)binario > T(n)secuencial$

$n(\log_2 \mathrm{A}\,(2\delta + 6 + \Delta) + \Delta + 5) + 3 > n \log_2 \mathrm{A}\,(4 + \delta + \Delta) + 3\mathrm{n} + 3$

$n(\log_2 \mathrm{A}\,(2\delta + 6 + \Delta) + \Delta + 5) > n(\log_2 \mathrm{A}\,(4 + \delta + \Delta) + 3)$

$\log_2 \mathrm{A}\,(2\delta + 6 + \Delta) + \Delta + 2 > \log_2 \mathrm{A}\,(4 + \delta + \Delta)$

$\Delta > -2 - 2\log_2 \mathrm{A} - \delta \log_2 \mathrm{A}$

Por esto, en el caso de las inyecciones inferenciales basadas en tiempo, esto es real para valores de retardo de tiempo superiores a una ecuación que siempre será negativa: $\Delta > -2 - 2\log_2 \mathrm{A} - \delta \log_2 \mathrm{A}$. Esto siempre será cierto ya que $\Delta > 0$

- **Conclusión**

El tiempo de ejecución del algoritmo bit a bit es menor que el tiempo empleado por el algoritmo binario para las inyecciones inferenciales no basadas en tiempo y para las inyecciones basadas en tiempo.

3.5 Comprobación empírica

En la revisión de literatura, se constató que se usan tiempos aleatorios para mostrar los ejemplos de retardo en la inyección (Δ), así (W. G. J. Halfond et al., 2008) usa 5 segundos sin indicar la relación con el tiempo de respuesta de la página, al igual que en (Anley, 2002b), en (Chandrashekhar, Mardithaya, Thilagam, & Saha, 2012a) se usa 10 segundos.

(Clarke & Alvarez, 2009) indica que si el tiempo de carga de la página es de 50 ms, un tiempo de retardo inyectado de 30s provee un periodo de tiempo suficiente para diferenciar la inyección del tiempo de respuesta normal. En otras palabras, si $\delta = 50\text{ms}$ entonces $\Delta = 30\text{s}$.

Con base en las demostraciones anteriores, para las inyecciones no basadas en tiempo, el mejor algoritmo es la extracción bit a bit, seguido de la búsqueda binaria y finalmente, la búsqueda secuencial. Para el caso de las inyecciones basadas en tiempo, en la tabla 13 se encuentra un resumen que permite identificar cual es el algoritmo que usa un menor tiempo para la extracción de información de acuerdo con el tiempo usado en la inyección.

Tabla 13 *Comparación de algoritmos por tiempo de respuesta en inyecciones inferenciales basadas en tiempo*

Valores Δ	Secuencial	Binario	Bit a bit
$< \delta(\frac{\mathrm{A}}{\log_2 \mathrm{A}} - 2)$	3	2	1
$> \delta(\frac{\mathrm{A}}{\log_2 \mathrm{A}} - 2)$ $\leq \frac{\mathrm{A}(\delta+2)-\log_2 \mathrm{A}(\delta+4)}{\log_2 \mathrm{A}-1}$	2	3	1
$> \delta(\frac{\mathrm{A}}{\log_2 \mathrm{A}} - 2)$ y $> \frac{\mathrm{A}(\delta+2)-\log_2 \mathrm{A}\,(\delta+4)}{\log_2 \mathrm{A}-1}$	1	3	2

Fuente: Elaboración propia

Teniendo en cuenta la tabla 13, se puede ver que el algoritmo de búsqueda secuencial podría llegar a tener mejor tiempo de extracción usando un tiempo de inyección mayor a la formulación indicada, relacionada directamente con el número de caracteres UNICODE a comparar y el tiempo de respuesta del sitio web.

De acuerdo con la metodología, se realizó un ejercicio empírico para comprobar los resultados teóricos anteriores. Se usaron 500 caracteres aleatorios (n=500), en una misma maquina bajo las mismas condiciones de procesamiento, memoria RAM, velocidad de lectura de disco. Para la extracción de estos caracteres, se tomaron los primeros 255

caracteres UNICODE (A=255) y se modificaron durante el ejercicio los parámetros definidos para comprobar los cálculos realizados anteriormente.

En la tabla 14 se encuentra la comprobación empírica de los tiempos de ejecución, comprobando a satisfacción los cálculos realizados.

Tabla 14 *Comprobación empírica de los algoritmos en inyecciones inferenciales no basadas en tiempo*

No basados en tiempo	Tiempos de ejecución
Secuencial	3.98 secs
Búsqueda binaria	0.815 secs
Bit a bit	0.376 secs

Fuente: Elaboración propia

Para la comprobación empírica de los algoritmos basados en tiempo, se realizó el mismo ejercicio anterior, pero con un sitio web que tuvieran exclusivamente ese tipo de vulnerabilidad. Los resultados de este ejercicio se observan en la tabla 15.

Tabla 15 *Comprobación empírica de los algoritmos en inyecciones inferenciales basadas en tiempo*

Valores Δ (Formulado)	Valores Δ (real)--	Secuencial	Binario	Bit a bit
$< 30\delta$	$(29\delta)\delta$=0.005, 0,18	32.62 s	27,15 s	24,63 s
$> 30\delta$ y $\leq 35\delta + 68$	$(34\delta)\delta$=0.006,	34.17 s	30.211 s	26.0484 s
$> 30\delta$ y $> 35\delta + 68$	$(36\delta)\delta$=0.0057, 0.2	34.69 s	29.8 s	27.434 s
$> 30\delta$ y $> 35\delta + 68$	$(48\delta)\delta$=0.0057	37.629 s	38.05 s	36.792 s
Aleatorio según bibliografía	1 s	56.478 s	139.0555 s	139.010 s

Fuente: Elaboración propia

En la primera columna de esta tabla, se detalla el valor teórico para el ejercicio empírico. Por ejemplo, el primero de estos ejercicios, es inyectando un retraso de tiempo de menos de 30 veces el tiempo de respuesta del sitio web. En la segunda columna, está el tiempo real inyectado en el ejercicio. Tomando el ejemplo anterior, se inyecta 0,18 segundos, el cual se calculó de la siguiente manera: 29 veces el tiempo de carga del sitio web, que fue de 0,005 segundos. Con ese valor calculado, se procedió a ejecutarla extracción de los caracteres con cada algoritmo. Se detallan en las siguientes tres columnas los tiempos empleados por cada uno de estos. Como puede observarse, tal y cómo se había calculado de manera teórica, el algoritmo bit a bit es el que emplea menos tiempo para la extracción de la información.

3.6 Tiempo de respuesta del sitio web

Durante la realización del ejercicio empírico, se notó la necesidad de especificar las diferentes maneras de calcular el tiempo de respuesta de un sitio web debido a que no se encontró en la revisión bibliográfica documentación al respecto. Este valor es denominado por la variable δ en este documento y es el tiempo que toma el servidor web para recibir y procesar una solicitud para una página de parte del cliente web (Latencia). Se ve influido por el RTT (Round Trip delay Time) de la conexión, que es el tiempo que tarda una petición de red en ir a su destino y volver, los tiempos implícitos en el protocolo TCP/IP (Como el Three-Way Handshake). (Naik & Jenkins, 2016)

Por lo anterior, los valores que adopta son completamente variables al ser diferentes en cada petición del sitio web, encontrándose con picos indeterminados y que incrementan la complejidad para la implementación de los algoritmos antes mencionados. Para calcular este valor, esta investigación determinó los siguientes métodos:

1. Realizar un número alto de peticiones al sitio web antes de iniciar el proceso de extracción, calcular el tiempo total de respuesta de cada una y usar el promedio. Al establecer un promedio, el valor será más acertado en la medida que el valor de las peticiones usadas para su cálculo sea mayor. Sin embargo, con el paso del tiempo, el servidor web puede degradarse, o los tiempos asociados factores de red pueden cambiar, lo que ocasiona que este valor tenga que recalcularse pronto.

2. Realizar un número alto de peticiones al sitio web, calcular el tiempo total de respuesta de cada uno y usar el mayor valor. Al usar el mayor valor, se está considerando el peor de los casos. Sin embargo, este valor puede ser muy alto comparado con el promedio, lo que incrementa significativamente los tiempos de extracción y, aun así, no hay garantía qué durante la extracción, no haya una latencia mayor.
3. Realizar un llamado al sitio web y calcular su tiempo justo antes de realizar alguna petición de extracción. Este método es una buena opción, ya que existe una alta probabilidad que la latencia sea similar en dos llamados realizados con un mínimo de tiempo entre ambos. Sin embargo, en la extracción de información, la cantidad de llamados al sitio web se duplica, por lo que se incrementa el tiempo de extracción.

Para establecer el método más efectivo, se realizó un experimento comparativo con el algoritmo secuencial y, de esta manera, determinar de manera empírica la mejor manera para realizar este cálculo tomando en cuenta su impacto en el tiempo y en el surgimiento de errores. Los resultados de este experimento se pueden ver en la tabla 16.

En esta tabla, se asignaron valores mínimos de inyección para ser ejecutados por el algoritmo secuencial. Estos tiempos se calcularon de las tres maneras mencionadas anteriormente: promedio de 200 peticiones, tiempo máximo de 200 peticiones y 1 petición antes de la inyección. Esto se hizo con el fin de observar la precisión del algoritmo con cada uno de estos métodos. El tiempo máximo reduce los falsos positivos, generando menos errores, pero influye radicalmente en el tiempo total empleado para la extracción. El cálculo antes de la inyección genera la mayor cantidad de errores aún con tiempo de inyección altos. La opción promedio es la que mejor relación costo/beneficio tiene: menor tiempo empleado en la extracción comparado con la exactitud de la información extraída.

Por lo tanto, en las pruebas realizadas tanto la opción de promedio cómo la opción de máximo son las mejores opciones para el cálculo de la latencia. Para los ejercicios empíricos se usó el cálculo a través del promedio para reducir el tiempo empleado en la prueba de penetración sin desmejorar la exactitud de la misma.

Tabla 16 *Efecto del cálculo del tiempo de respuesta sobre el algoritmo secuencial*

Valores Δ	Secuencial *De un total de 200 peticiones		
	AVG*	MAX*	Antes
2δ	16.1s	23.95s	45.94s
	25 error	16 error	23 error
3δ	26.9s	46.96s	69.91s
	24 error	7 errores	18 error
4δ	39.2 s	42.96 s	76.8 s
	10 error	1 error	12 error
5δ	37.4 s	32.25 s	99.70 s
	2 error	5 errores	9 errores
10δ	42.07s	48.57s	95.09 s
	1 error	1 error	8 errores
15δ	38.67s	35.57s	106.17s
	1 error	1 error	1 error
16δ	48.04 s	39.11 s	112.43 s
	0 error	0 error	0 error

Fuente: Elaboración propia

3.7 Retraso de tiempo a inyectar

En esta tesis se usa la variable Δ para indicar el tiempo inyectado en la consulta modificada. En la revisión bibliográfica realizada en esta investigación sobre las inyecciones SQL inferenciales no se encuentra la manera de calcular el valor para la inyección de tiempo. Antes de inyectar un retraso de tiempo para forzar un cambio de comportamiento, es necesario escoger cuidadosamente el valor de tiempo óptimo. Un valor significativamente

alto, puede conllevar una demora innecesaria en la extracción. Y un valor significativamente bajo, podría generar que en una consulta determinada el valor inyectado sea menor o igual a la latencia del sitio web ($\Delta \leq \delta$), lo que genera un falso positivo generando inexactitud en los datos extraídos.

3.7.1 Cota inferior del retraso de tiempo a inyectar

El tiempo a inyectar debe cumplir con la siguiente desigualdad: Tiempo a inyectar debe ser mayor que el tiempo de respuesta del sitio web ($\Delta > \delta$). Teniendo en cuenta que el tiempo de respuesta (δ) es variable y que para cada extracción debe realizar un número indeterminado de peticiones, debe cumplirse lo siguiente:

$$\Delta > \delta 1,\ \Delta > \delta 2, \Delta > \delta 3, \Delta > \delta 4 \ldots \Delta > \delta \mathrm{m}$$

Es decir, para todas las *m* peticiones realizadas al sitio web, el valor inyectado debe ser mayor.

Debido a que no es posible conocer el tiempo exacto de latencia antes de realizar la petición, el tiempo a inyectar debe ser lo suficientemente mayor para que tenga tolerancia al error. Por tanto, siempre que inyectemos un retardo de tiempo existirá un riesgo de generar un falso positivo.

Para esto, en esta investigación se define un nuevo índice, llamado índice de precisión del algoritmo (IPA), siendo calculado como la cantidad de caracteres correctamente extraídos sobre la cantidad total de caracteres a extraer. Para ilustrar el uso de este índice, se elaboró la tabla 17 a partir de ejercicios empíricos con tiempos de inyección diferentes. Para este ejercicio, se calculó el valor del tiempo de cargue del sitio web (δ) antes de cada inyección.

De acuerdo con los experimentos similares realizados y evidenciados en la tabla 17, es mejor usar como tiempo a inyectar un valor superior a 16δ. Para mayor exactitud de acuerdo a la latencia del sitio web, es mejor realizar un análisis similar para determinar la cota inferior para cada caso particular. Igualmente, puede notarse en dicho experimento que el algoritmo bit a bit responde mejor ante valores bajos de inyección.

Tabla 17 *Índice de precisión del algoritmo IPA para valores mínimos Δ*

Valores Δ	Secuencial	Binario	Bit a bit
2δ	3.33%	50%	66.67%
3δ	13.33%	80%	96.66%
4δ	33.33%	96.67%	100%
5δ	53.33%	80%	93.33%
10δ	93.33%	96.66%	96.66%
15δ	96.66%	100%	100%
16δ	100%	100%	100%

Fuente: Elaboración propia

3.7.2 Cota superior

Teniendo claro que cualquier valor superior a los imprevistos en la latencia del sitio web puede servir para la extracción de la información, es necesario delimitar los valores óptimos de acuerdo a cada algoritmo usado con el fin de no emplear más tiempo del necesario en la extracción de la información. Por tal razón, la comparación realizada en la sección 3.4 determina las cotas superiores óptimas.

3.8 Optimizaciones frecuentes a los algoritmos inferenciales

Los algoritmos analizados pueden ser optimizados de varias maneras. Durante esta investigación se determinaron tres optimizaciones comunes, siendo posible usar un solo método o varios en una misma implementación, dependiendo del algoritmo. En la tabla 18 puede verse un resumen de estas optimizaciones que se explican a continuación.

Tabla 18 *Optimizaciones frecuentes a los algoritmos inferenciales*

	Secuencial	Binario	Bit a bit
Diccionario	Si	Si	No
Ordenamiento por probabilidad	Si	No	Si
Paralelizar	No	No	Si

Fuente: Elaboración propia

3.8.1 Uso de diccionarios

En la medida que se van extrayendo caracteres de la base de datos y si se tiene identificado el lenguaje de la base de datos, puede usarse un diccionario para generar el siguiente carácter a probar. Por ejemplo, en el idioma inglés, si se han extraído dos caracteres, cómo Th, se puede usar un diccionario que permita evitar intentos cómo Thz, y concentrarse en intentos cómo The, Tha, Thi, Tho, Thu, con lo cual es probable encontrar rápidamente el siguiente carácter. Esta optimización añade mayor tiempo de ejecución al algoritmo debido a la consulta que realiza sobre el diccionario después de la extracción de cada carácter, pero se ve recompensado en la disminución de las consultas modificadas enviadas al sitio web.

3.8.2 Ordenamiento por probabilidad de ocurrencia

Conociendo el idioma de la base de datos, es posible ordenar los caracteres de acuerdo a la probabilidad de ocurrencia. Así, suponiendo que el carácter más común en el idioma inglés es la letra ‘e’, ese carácter sería el primero en la lista ordenada por ocurrencia. En el algoritmo bit a bit, también es posible ordenar el bit a preguntar de acuerdo a un cálculo de probabilidad.

3.8.3 Paralelizar

Teniendo en cuenta que cada carácter a extraer de la base de datos puede hacerse de manera independiente, se pueden implementar una arquitectura multi-hilos, donde cada hilo se dedique a un carácter en particular. Aunque todos los algoritmos pueden ser paralelizados de esta manera, en el algoritmo bit a bit es posible incluso paralelizar cada bit, ya que la extracción de uno es independiente a los otros.

4. Análisis y comparación de las herramientas para la evaluación de inyecciones SQL inferenciales

Después de realizar una búsqueda sistemática en diferentes fuentes académicas, las cuales en detallaron en la sección 2.5, se realizó una lista de las herramientas encontradas la cual se encuentra en la tabla 19. Con cada una de ellas se realizaron los siguientes pasos:

1. Buscar en la documentación y sitios web asociados la última versión de la herramienta disponible y el sistema operativo compatible.
2. Determinar si tiene soporte actual a través de la respuesta en foros, sitios web y último año de versión disponible. Para las herramientas: Absinthe, Pixy, SQL Brute, Havij, Pangolin, SQID, SAFE3 SQL INJECTOR, Shoryuken, PRIAMOS y BSQLBF (10 herramientas), no se cuenta con sitio web activo ya que los enlaces principales están rotos.
3. Realizar la instalación de las herramientas en los sistemas operativos compatibles. Las herramientas SQLIX, SQLNinja, BBQSQL, Squeeza, SQID, Shoryuken y ISR-SQLGet (7 herramientas) fueron instalados en el sistema operativo Backtrack 5.0, un sistema operativo que estaba vigente en el año de lanzamiento de estas herramientas. La herramienta V3N0M viene instalada por defecto en el sistema operativo Kali. La herramienta SQLIX requiere librerías desactualizadas que no se consiguen en Internet por los canales oficiales de los lenguajes de programación. La herramienta SQID ya no se encuentra en el sitio oficial de Ruby y fue necesario buscarla en archivos no oficiales de Internet. La herramienta BlindCanSeeQL se encuentra documentada con fecha de creación en el año 2015, pero aún no tiene un reléase oficial.

Para cada herramienta, se documentó su nombre comercial, licencia, año de la última versión, lenguaje en que se desarrolló, bases de datos que soporta para la extracción, si tiene soporte activo o no y los sistemas operativos donde funcionan. Esta información también se documentó en la tabla 19 y las abreviaturas se encuentran definidas en la lista correspondiente al inicio de este documento.

Tabla 19 *Lista de herramientas para explotación del riesgo inyecciones SQL*

Nombre Comercial	**Licencia**	**Año de la última versión**	**Lenguaje**	**Base de datos soportadas**	**Soporte**	**Sistemas operativos**
Absinthe	GNU GPL	2007	.NET	MS,ORA, PG	No	Win7
Marathon Tool	Ms-PL	2008	.NET	MS, ORA, MY,1+	No	Win10
SQLiX	Creative Commons Attribution 3.0	2006	PERL	MS, MY	No	Linux
Acunetix 11 Trial	Shareware 14 days	2017	.NET	MS, ORA, PG, MY,+	Si	Win10
Sqlmap	GNU GPL	2017	Python	MS, ORA, PG, MY,12+	Si	Win, Linux, MAC
Wapití	GNU GPL	2013	Python	MS, MY	No	Win10
Paros	Creative Commons Attribution	2006	Java	MS	No	Win7
Pixy	Open Source	2007	Java	MY	No	Win10
BSQL Hacker	GNU GPL	2008	.NET	MS, ORA, MY	No	Win7

SQLBrute - SQL Injection Brute Forcer	Reciprocal Public License 1.5	2006	Python	MS,ORA	No	Win10
SQLNinja	GNU GPL	2012	Perl	MS	No	Linux
BBQSQL	BSD Licensing	2014	Python	MY	No	Linux
Squeeza	GNU GPL	2012	Ruby	MS, MY	No	Linux
Havij	Shareware	2013	.NET	MS	No	Win7
SQL Power Injector	Clarified Artistic license	2014	.NET	MS, ORA, MY,2+	No	Win10
BlindCanSeeQL 2015	No released	No released	.NET	MS	No	Sin version
Pangolin	Shareware	2012	Unknow	MS, ORA, PG, MY, 5+	No	Win7
DarkMysqli	GNU GPL	2009	Python	MY	No	Win10
SQID - SQL Injection digger	GNU GPL	2007	Ruby	MS, ORA, MY	No	Linux
Safe3 SQL Injector	Open Source	2011	Unknow	MS, ORA, PG, MY, 4+	No	Win7
The Mole	GNU GPL	2012	Python	MS, ORA, PG, MY	No	Win10
V3n0m	GNU GPL	2017	Python	MS, ORA, MY, 1+	Si	Linux
Shoryuken - SQL Injection Takeover	GNU GPL	2013	Linux bash	MS, MY	No	Linux
FJ-Injector Framework	GNU GPL	2007	C++	MS, ORA, PG, MY	No	Win7
ISR-SQLGet	GNU GPL	2007	Perl	MS, ORA, PG, MY, 17+	No	Linux

Priamos	Freeware	2007	.NET	MS	No	Win10
BSQLBF	BSD Licensing	2009	Perl	MS, ORA, PG, MY	No	Win10

Fuente: Elaboración propia

De acuerdo con la tabla 19, solo el 11% de las herramientas tiene un licenciamiento pago, aunque tienen versiones demo que permiten realizar su evaluación. También es notable el corto tiempo de vida de las herramientas analizadas: Durante los años 2004 a 2007, se crearon el 77.7% de las 27 herramientas analizadas. Sin embargo, de ese porcentaje, solo una herramienta (SQLMAP) tiene soporte activo vigente y versiones compatibles con las bases de datos más recientes. La herramienta V3n0m es la otra herramienta que cuenta con licenciamiento libre y soporte vigente, aunque su año de creación es más reciente a SQLMAP.

La compatibilidad con las bases de datos es otro factor que diferencia una herramienta de otra. La herramienta ISR-SQLGET tiene en su documentación compatibilidad con 21 bases de datos diferentes, esto es comprensible ya que es una herramienta manual donde el conocimiento de la base de datos recae en el usuario y no en la herramienta. La siguiente herramienta con una mayor compatibilidad de base de datos es SQLMAP con un total de 12 bases de datos compatibles, esto es resultado de los años de madurez de esta herramienta. El resto de las herramientas es compatible con menos, siendo la media entre 3 y 4 bases de datos. De las 27 herramientas, 24 son compatibles con Microsoft SQL Server, 20 con MySQL, 15 con ORACLE, 9 con POSTGRES y 6 con ACCESS. Las 27 herramientas son compatibles con Microsoft SQL Server o MySQL, por lo que un entorno de prueba con estas dos bases de datos se pueden analizar todas las herramientas.

Con base en la información anterior, se realizó un análisis comparativo. Aunque las herramientas en cuestión son para la evaluación del riesgo de las inyecciones SQL, no todas evalúan el riesgo de inyecciones inferenciales. Esta investigación limita la evaluación a dicho riesgo, analizando el tiempo empleado por cada herramienta de la manera indicada por la metodología.

4.1 Pasos para la evaluación empírica de la herramienta

Con el objetivo de determinar cuál herramienta usar para la evaluación del riesgo de inyecciones SQL inferenciales, se fijaron dos objetivos:

1. Conocer cuáles son los algoritmos implementados por cada herramienta para la evaluación del riesgo de inyecciones inferenciales.
2. Realizar una comparación del tiempo de ejecución para establecer cuál es la herramienta que evalúa el riesgo en menos tiempo.

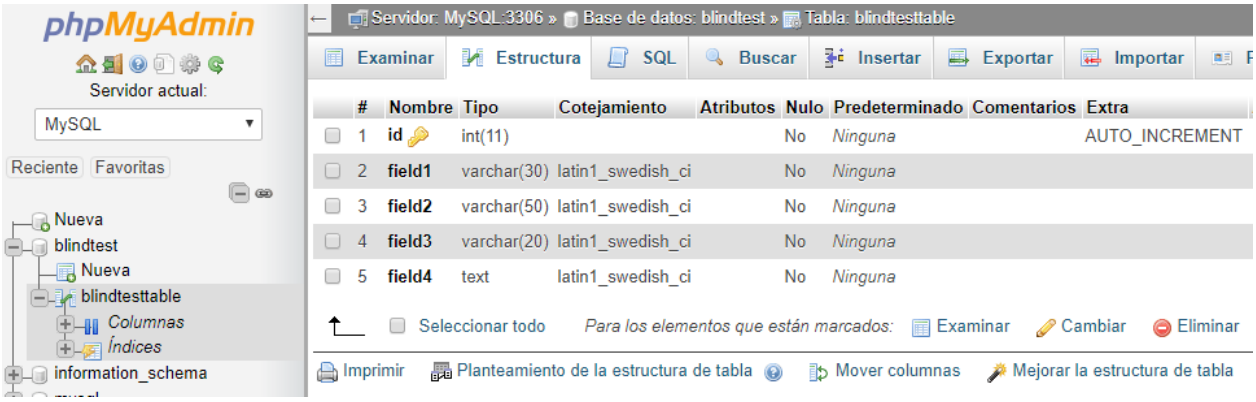
Con este fin, se analizaron las bases de datos compatibles a cada herramienta. El análisis empírico se realizó con las bases de datos SQL Server y MySQL. Para conocer el algoritmo y el tiempo usado de cada herramienta se siguieron los siguientes pasos:

1. Crear una base de datos con 1 sola tabla y 5 campos, alimentada con dos registros, uno con 500 caracteres aleatorios y otro con 500 caracteres en palabras del idioma inglés.
2. Elaborar cuatro páginas web susceptibles al riesgo de inyecciones inferenciales conectados a dicha aplicación: por cada sub clasificación del riesgo (basadas en tiempo y no-tiempo) y por cada base de datos identificada (SQL SERVER y MYSQL).
3. Analizar los logs asociados en el servidor de aplicaciones después de la ejecución de cada herramienta. Es necesario analizar cada petición realizada al servidor web e identificar el algoritmo utilizado en cada una.
4. Basado en el análisis anterior, identificar el algoritmo, analizar los tiempos de ejecución de cada herramienta para cada tipo y determinar la mejor implementación para la evaluación del riesgo.

4.2 Preparación del entorno de prueba

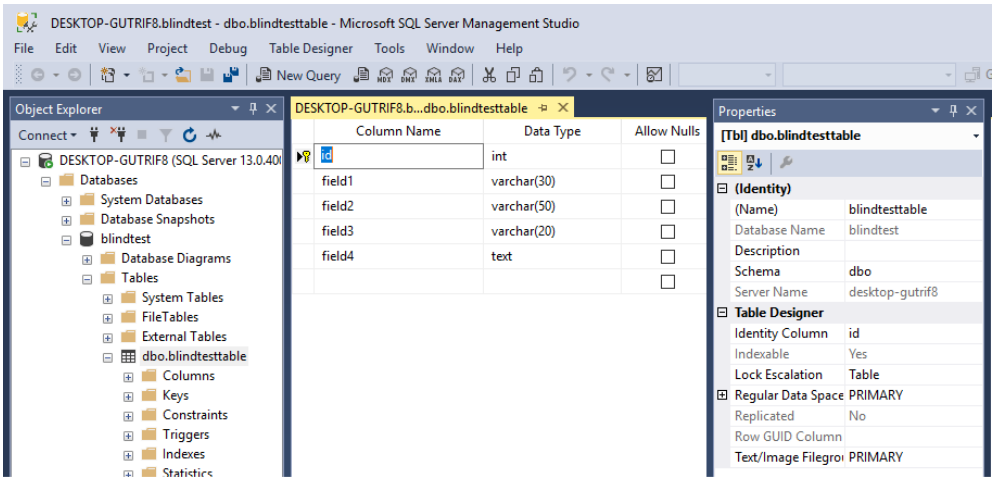
Para la realización de la prueba de evaluación fue necesario crear una máquina virtual VMWARE con 2 Gb de RAM y 1 procesador Inter Core I7-6600U CPU@ 2.6 Ghz, corriendo un sistema operativo Windows 10 Enterprise a 64 bits. Sobre esa máquina virtual, se instalaron dos motores de base de datos: MySQL Version 5.7.19 y SQL Server versión 13.0.4001.0. Para estas bases de datos se corrieron los scripts que crean la que será base de datos de prueba. Sobre estas bases de datos, ambas llamadas blindtest, se creó una tabla llamada blindtesttable. En la figura 7 se encuentra la estructura creada en MYSQL y en la figura 8 se encuentra la misma estructura implementada en SQL Server.

Figura 7 *Estructura de la base de datos MySQL de prueba*



Fuente: Elaboración propia

Figura 8 *Estructura de la base de datos SQL Server de prueba*



Fuente: Elaboración propia

Además de lo anterior, se instaló un solo servidor web que se conectó a ambas bases de datos. El servidor web es un servidor apache versión 2.4.27 con PHP versión 7.0.23 y MySQL Version:5.7.19. Las librerías que tiene el apache implementadas son: apache2handler, bcmath, bz2, calendar, com_dotnet, Core, ctype, curl, date, dom, exif, fileinfo, filter, gd, gettext, gmp, hash, iconv, imap, intl, json, ldap, libxml, mbstring, mcrypt, mysqli, mysqlnd, openssl, pcre, PDO, pdo_mysql, pdo_sqlite, pdo_sqlsrv, Phar, Reflection, session, SimpleXML, soap, sockets, SPL, sqlite3, sqlsrv, standard, tokenizer, wddx, xdebug, xml, xmlreader, xmlrpc, xmlwriter, xsl, Zend OPcache, zip, zlib.

Con respecto a las maquinas empleadas, se ejecutó la herramienta en un equipo portátil con procesador Intel Core i7 a 2.6 Ghz Gb, 16 Gb de RAM, disco de estado sólido de 512 Gb y sistema operativo Windows 10 Pro. Este equipo se conectó al otro a través de un Switch Cisco Catalyst 2970 y cableado AMP categoría 6A. La máquina virtual se ejecutó sobre un servidor Dell PowerEdge M910 con dos procesadores Intel Xeon CPU E7-2870 @ 2.40GHz con 196 Gb de RAM, Disco duro SCSI 271,50 GB, adaptador de red 10/100/1000 Mbps con sistema operativo VMWare ESXi -5.5.0 standard.

Sobre este servidor web, se crearon 4 páginas web:

1. Errorbasedtestmssql: Pagina web vulnerable a inyecciones SQL inferenciales no basadas en tiempo y compatible con SQL Server (Figura 9).
2. Errorbasedtestmysql: Pagina web vulnerable a inyecciones SQL inferenciales no basadas en tiempo y compatible con MySQL Server (Figura 10).
3. Timebasedtestmssql: Pagina web vulnerable a inyecciones SQL inferenciales basadas en tiempo y compatible con SQL Server (Figura 11).
4. Timebasedtestmysql: Pagina web vulnerable a inyecciones SQL inferenciales basadas en tiempo y compatible con MySQL Server (Figura 12).

Para la realización de este código fuente, se tomaron las pautas establecidas en la documentación encontrada (Slaviero, 2012). Las figuras 9 y 10 difieren en la manera de conexión a la base de datos, las cuales en la figura 9 son usando las librerías de SQL Server para PHP y en la figura 10, las librerías de MySQL para el mismo lenguaje. Igual diferencia existe entre el código de las figuras 11 y el de la figura 12.

Además, las diferencias entre las figuras 9 y 10 (no basadas en tiempo) con las figuras 11 y 12 (basadas en tiempo), son la existencia de mensajes de error y otra información arrojada de la base datos y mostrada en la página web, información que se aprovecha en el caso de las inyecciones inferenciales no basadas en tiempo.

Figura 9 *Archivo Errorbasedtestmssql.php*

```php
<?php
error_reporting(0);
if(isset($_GET['id'])){
    $id = $_GET['id'];
    // Puesto que no se han especificado UID ni PWD en el array  $connectionInfo,
    // La conexión se intentará utilizando la autenticación Windows.
    $serverName = "localhost";
    $connectionOptions = array("Database"=>"blindtest");
    $conn = sqlsrv_connect($serverName, $connectionOptions);
    if( $conn ) {
        echo "Conexión establecida.<br />";
    }
    else {
        echo "Conexión no se pudo establecer.<br />";
        die( print_r( sqlsrv_errors(), true));
    }
    $sql = "SELECT * FROM blindtesttable WHERE id = " . $id;
    $result = sqlsrv_query($conn,$sql);
    try {
        $result = sqlsrv_query($conn,$sql);
    }
    catch(Exception $e) {
    }
    $stmt = sqlsrv_query( $conn, $sql, $params);
        if( $stmt === false ) {
        }
    while( $row = sqlsrv_fetch_array( $stmt, SQLSRV_FETCH_ASSOC) )  {
        if($row['id']==$id)
            echo $row['fieldl'];
        else
            echo "Error en la consulta";
    }
    sqlsrv_free_stmt( $stmt);
}
?>
<html>
<head>
</head>
<body>
    <form action="errorbasedtestmssql.php">
        Enter ID: <input type="text" name="id"/>
        <input name="search" type="submit" value="Search">
    </form>
</body>
</html>
```

Fuente: Elaboración propia

Figura 10 *Archivo errorbasedtestmysql.php*

```php
<?php
error_reporting(0);
if(isset($_GET['id'])){
    $id = $_GET['id'];
    $link =  mysqli_connect('localhost', 'root', '','blindtest');
    if (!$link) {
        die('Conection Failed' . mysql_error());
    }
    $sql = "SELECT * FROM blindtesttable WHERE id = " . $id;
    try {
        print($sql);
        $result = mysqli_query($link,$sql);
    }
    catch(Exception $e) {
            print("errr");
    }
    while ($result && $row = mysqli_fetch_assoc($result))    {
        if($row['id']==$id)
            echo $row['fieldl'];
    }
    if($result)
        mysqli_free_result($result);
    mysqli_close($link);
}
?>
<html>
<head>
</head>
<body>
    <form action="errorbasedtestmysql.php">
        Enter ID: <input type="text" name="id"/>
        <input name="search" type="submit" value="Search">
    </form>
</body>
</html>
```

Fuente: Elaboración propia

Figura 11 *Archivo timebasedtestmssql.php*

```php
<?php
error_reporting(0);
if(isset($_GET['id'])){
    $id = $_GET['id'];
    // Puesto que no se han especificado UID ni PWD en el array  $connectionInfo,
    // La conexión se intentará utilizando la autenticación Windows.
    $serverName = "localhost";
    $connectionOptions = array("Database"=>"blindtest");
    $conn = sqlsrv_connect($serverName, $connectionOptions);
    if( $conn ) {
        $sql = "SELECT * FROM blindtesttable WHERE id = " . $id;
        $result = sqlsrv_query($conn,$sql);
        try {
            $result = sqlsrv_query($conn,$sql);
        }
        catch(Exception $e)
        {}
        $stmt = sqlsrv_query( $conn, $sql);
        while( $row = sqlsrv_fetch_array( $stmt, SQLSRV_FETCH_ASSOC) )  {
            if(strcmp($row['id'],$id)>=0)
                echo $row['fieldl'];
        }
        sqlsrv_free_stmt( $stmt);
    }
}
?>
<html>
<head>
</head>
<body>
    <form action="timebasedtestmssql.php">
        Enter ID: <input type="text" name="id"/>
        <input name="search" type="submit" value="Search">
    </form>
</body>
</html>
```

Fuente: Elaboración propia

Figura 12 *Archivo timebasedtestmysql.php*

```php
<?php
error_reporting(0);
if(isset($_GET['id'])){
    $id = $_GET['id'];
    $link =  mysqli_connect('localhost', 'root', '','blindtest');
    if (!$link) {
        die('Conection Failed' . mysql_error());
    }
    $sql = "SELECT * FROM blindtesttable WHERE id = " . $id;
    try {
        $result = mysqli_query($link,$sql);
    }
    catch(Exception $e)
    {}
    while ($result && $row = mysqli_fetch_assoc($result)) {
        if($row['id']==$id)
            echo $row['fieldl'];
        else
            echo "Error en la consulta";
    }
    if($result)
        mysqli_free_result($result);
    mysqli_close($link);
}
?>
<html>
<head>
</head>
<body>
    <form action="timebasedtestmysql.php">
        Enter ID: <input type="text" name="id"/>
        <input name="search" type="submit" value="Search">
    </form>
</body>
</html>
```

Fuente: Elaboración propia

Con el objetivo de alimentar las bases de datos con información aleatoria, se creó el código mostrado en la figura 13. El método randomString retorna una cadena de caracteres aleatoria con caracteres con valores UNICODE entre 1 y 255. Y se llena cada uno de los

campos de la tabla creada de acuerdo a su longitud. Aunque este código alimenta la base de datos MySQL, se creó uno similar para la base de datos SQL Server cambiando los parámetros de conexión.

Figura 13 *Código para alimentar las bases de datos*

```php
<?php
    $link =  mysqli_connect("localhost", "root", '', "blindtest");
    if (!$link) {
        echo "Error: No se pudo conectar a MySQL." . PHP_EOL;
        echo "errno de depuración: " . mysqli_connect_errno() . PHP_EOL;
        echo "error de depuración: " . mysqli_connect_error() . PHP_EOL;
        exit;
    }
    function randomString($len) {
        $characters = "";
        for ($i = 1; $i < 256; $i++) {
            $characters = $characters . chr($i);
        }
        $charactersLength = strlen($characters);
        $randomString = '';
        for ($i = 0; $i < $len; $i++) {
            $rand = rand(0, $charactersLength - 1);
            $randomString .= $characters[$rand];
        }
        return $randomString;
    }
    $stmt = mysqli_prepare($link,"INSERT INTO blindtesttable(field1,field2,field3" .
                                 ",field4) VALUES (?,?,?,?)");
    if ( false===$stmt ) {
      die('prepare() failed: ' . htmlspecialchars(mysqli_error($link)));
    }
    for ($i = 0; $i < 1; $i++) {
        $field1 = randomString(30);
        $field2 = randomString(50);
        $field3 = randomString(20);
        $field4 = randomString(500);
        if(mysqli_stmt_bind_param($stmt,'ssss',$field1,$field2,$field3,$field4))
            mysqli_stmt_execute($stmt);
    }
    echo 'Connection Sucessfull';
    mysqli_close($link);
?>
```

Fuente: Elaboración propia

Con los pasos indicados en esta sección, se tienen dos motores de base de datos instalados, dos bases de datos creadas y alimentadas con datos aleatorios (una en cada motor) y cuatro páginas web que son vulnerables a los dos subtipos y compatibles con ambas bases de datos.

4.3 Desarrollo de la comparación empírica de las herramientas

El objetivo de la comparación es analizar el uso de las herramientas en la evaluación del riesgo de inyecciones SQL inferenciales. Existen varios tipos de herramientas: algunas identifican el riesgo y lo evalúan, otras solo lo identifican y otras solo lo evalúan.

4.3.1 Identificación del riesgo

La identificación consiste en tomar la URL del sitio web indicado por el usuario, analizar todas las entradas de este y, por cada entrada, realizar una batería de pruebas que permitan identificar si el sitio es vulnerable o no. El proceso de identificación también pudiera incluir el mismo análisis sobre las páginas internas y así de manera recursiva hasta no encontrar más.

De las herramientas analizadas, 4 (15%) analizan las entradas del sitio web e identifican si son vulnerables al riesgo de inyecciones SQL. Sin embargo, solo 3 de ellas (Acunetix, Wapití y V3n0m) encontraron la vulnerabilidad a las inyecciones inferenciales.

4.3.2 Evaluación del riesgo

El proceso de evaluación consiste en usar la entrada identificada en el proceso anterior y ejecutar inyecciones para la extracción de la información. Este proceso se realiza con el fin que el auditor determine la criticidad del riesgo en términos del impacto que puede tener en la organización. La mayoría de las herramientas solo realizan este paso, solicitando al usuario el ingreso de la posible entrada vulnerable.

De todas las herramientas analizadas, solo la herramienta SQLMAP tiene la opción para evaluar el riesgo inferencial basado en tiempo y no basado en tiempo, las herramientas Marathon Tool, SQLBrute - SQL Injection Brute Forcer,SQLNinja,BBQSQL,Squeeza y

V3N0M (6 herramientas, 22%) tienen la funcionalidad para evaluar las inyecciones basadas en tiempo y las herramientas Absinthe, BSQL Hacker, SQL Power Injector, PANGOLIN, DARKMYSQLI, SAFE3 SQL INJECTOR, THE MOLE, FJ-Injector Framework, BSQLBF 1.1 (9 herramientas, 33%) tienen la funcionalidad para evaluar las no basadas en tiempo.

Con respecto a las entradas solicitadas al usuario por parte de las herramientas para iniciar su ejecución, las herramientas Acunetix 11 Trial, Wapití y PRIAMOS (3 herramientas, 11%) necesitan solo la URL del sitio web, las herramientas Absinthe, Marathon Tool, SQLiX, Sqlmap, BSQL Hacker, SQLBrute - SQL Injection Brute Forcer, SQLNinja, BBQSQL, Squeeza, Havij, SQID - SQL Injection digger, Shoryuken - SQL Injection Takeover (12, 44%) requieren la URL y la entrada vulnerable, BSQLBF, FJ-Injector Framework y The Mole (3 herramientas, 11%) requieren la URL, la entrada vulnerable y un parámetro llamado NEEDLE, que consiste en el cambio del comportamiento de la página a buscar después de la inyección. Las otras 8 herramientas, requieren más parámetros, cómo el código fuente del sitio web, puertos y hasta archivos de configuración.

4.4 Resultados de la evaluación

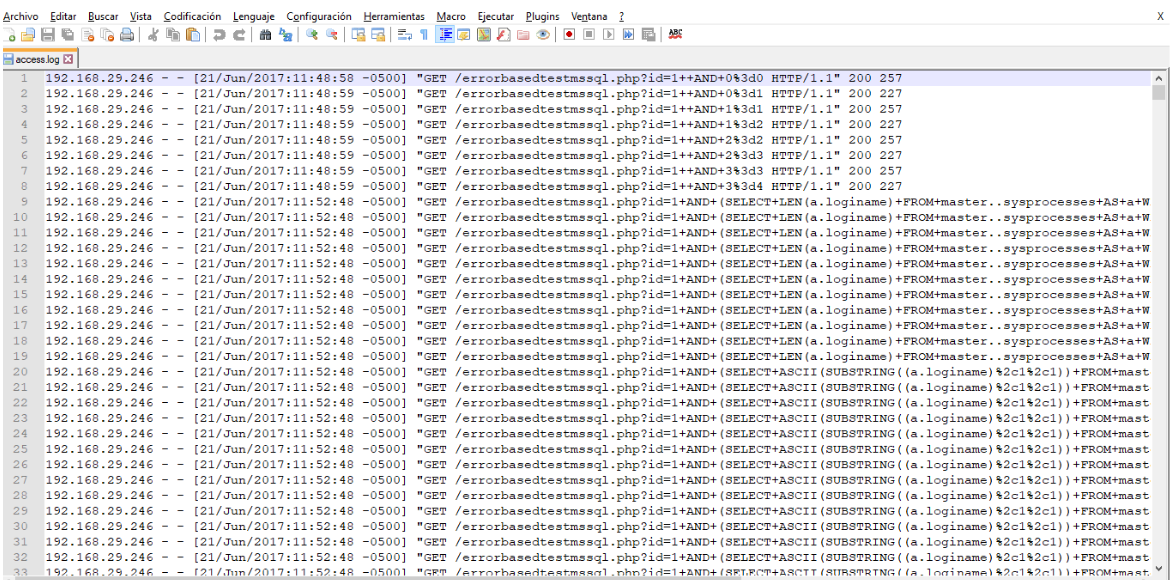
Al ejecutar las herramientas, se comprobó la extracción de la información. De las 27 herramientas, solo 5 (19%) extrajeron la información de manera correcta. Las herramientas Marathon Tool, SQLNinja, Squeeza, Havij, SQL Power Injector, Pangolin, DarkMysqli, Safe3 SQL Injector, FJ-Injector Framework y BSQLBF (37%) no son compatibles con las versiones de Mysql 5.0 o superior, ni SQL Server 2016 por lo que extrajeron datos de manera parcial. La herramienta Acunetix no permite la extracción en su versión demo y las herramientas SQLiX, Wapití, V3nom, BBQSQL, Paros, Pixy, SQID - SQL Injection digger, Shoryuken - SQL Injection Takeover, ISR-SQLGet, Priamos (26%) no tienen la opción de extracción de información o la opción no funciona.

Las herramientas Acunetix 11 Trial, Sqlmap, Wapití, Paros, Pixy, BSQL Hacker, BBQSQL y V3n0m (8, 29%) funcionan de manera automatizada, se indican los parámetros necesarios y la herramienta inicia automáticamente la extracción.

4.5 Algoritmos usados por las herramientas

Después de la extracción, se analizó el LOG del servidor web apache llamado Access.log. En la figura 14 se encuentra un ejemplo de cómo se ven las peticiones de las herramientas en dicho sitio.

Figura 14 *Log del servidor después de la ejecución de la herramienta Absinthe*




```
192.168.29.246 - - [21/Jun/2017:11:48:58 -0500] "GET /errorbasedtestmssql.php?id=1++AND+0%3d0 HTTP/1.1" 200 257
192.168.29.246 - - [21/Jun/2017:11:48:59 -0500] "GET /errorbasedtestmssql.php?id=1++AND+0%3d1 HTTP/1.1" 200 227
192.168.29.246 - - [21/Jun/2017:11:48:59 -0500] "GET /errorbasedtestmssql.php?id=1++AND+1%3d1 HTTP/1.1" 200 257
192.168.29.246 - - [21/Jun/2017:11:48:59 -0500] "GET /errorbasedtestmssql.php?id=1++AND+1%3d2 HTTP/1.1" 200 227
192.168.29.246 - - [21/Jun/2017:11:48:59 -0500] "GET /errorbasedtestmssql.php?id=1++AND+2%3d2 HTTP/1.1" 200 257
192.168.29.246 - - [21/Jun/2017:11:48:59 -0500] "GET /errorbasedtestmssql.php?id=1++AND+2%3d3 HTTP/1.1" 200 227
192.168.29.246 - - [21/Jun/2017:11:48:59 -0500] "GET /errorbasedtestmssql.php?id=1++AND+3%3d3 HTTP/1.1" 200 257
192.168.29.246 - - [21/Jun/2017:11:48:59 -0500] "GET /errorbasedtestmssql.php?id=1++AND+3%3d4 HTTP/1.1" 200 227
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+LEN(a.loginame)+FROM+master..sysprocesses+AS+a+W
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
192.168.29.246 - - [21/Jun/2017:11:52:48 -0500] "GET /errorbasedtestmssql.php?id=1+AND+(SELECT+ASCII(SUBSTRING((a.loginame)%2c1%2c1))+FROM+mast
```

Fuente: Elaboración propia

En esta investigación, se analiza el archivo resultante para inferir cuál es el algoritmo usado por cada herramienta. En total, el 37% de las herramientas usan el algoritmo de búsqueda binaria, el 41% no realizó peticiones al sitio web con inyecciones SQL debido a que no tenía la funcionalidad o no funcionaba con la versión actual de la base de datos, 7% usó la búsqueda secuencial con optimización, el 11% usó el algoritmo bit a bit y solo la herramienta SQLMAP tiene una implementación de los tres algoritmos con diferentes optimizaciones que se configuran desde el inicio de la ejecución si el usuario así lo desea.

La optimización más común es el ordenamiento de caracteres a buscar por frecuencia de aparición.

Los resultados de este trabajo se encuentran resumidos en la tabla 20 y las siglas usadas en esta tabla se encuentran al inicio de este documento. Cuando en la columna *entradas* se encuentra un número y el signo más, significa que, aparte de las entradas mencionadas, la herramienta exige más, si se indica 2+ significa 2 entradas más y así sucesivamente.

Tabla 20 *Análisis de las herramientas sobre el riesgo de inyecciones SQL inferenciales*

Herramienta	**Identificación**	**Extracción satisfactoria**	**Algoritmo**	**Técnicas**	**Optimización**	**Entradas**	**Operación**
Absinthe	No	Si	BS	NTB	-	URL+VI	Manual
Marathon Tool	No	No, solo para bases de datos en desuso	BS	TB	-	URL+VI	Manual
SQLiX	No	No	No	No	No	URL+VI	Manual
Acunetix 11 Trial	Si	No incluido en la versión de prueba	No incluido en la versión de prueba	No incluido	Desconocido	URL	Automático
Sqlmap	No	Si	3 algoritmo optimizada	TB, NTB	Probabilidad	URL+VI	Automático
Wapití	Si	No	No	No	No	URL	Automático

Paros	No	No	No	No	No	URL+VI+2+	Automático
Pixy	No	No	No	No	No	Código fuente	Automático
BSQL Hacker	No	Si	BS	NTB	-	URL+VI	Automático
SQLBrute - SQL Injection Brute Forcer	No	Si	SS Optimizado	TB	Hilos + Probabilística	URL+VI	Manual
SQLNinja	No	Parcial, solo metadato	SS Optimizado	TB	Probabilística	URL+VI	Manual
BBQSQL	No	Parcial	SS Optimizado	NTB	Probabilística	URL+VI+8+	Automático
Squeeza	No	No, solo para versiones en desuso	Bit a Bit	TB	-	URL+VI	Manual
Havij	No	No, solo para versiones en desuso		No	-	URL+VI	Manual
SQL Power Injector	No	No, solo para versione	BS	NTB	-	URL+VI+Nee+4+	Manual

		s en desuso					
BlindCanSeeQL 2015	No	Teórico	BS		Diccionario		Sin lanzamiento oficial
Pangolin	No	No, solo para versiones en desuso	BS	NTB		URL+VI+2+	Manual
DarkMysqli	No	No, solo para versiones en desuso	BS	NTB		URL+VI+Nee+1+	Manual
SQID - SQL Injection digger	No	No	No	No		URL+VI	Manual
Safe3 SQL Injector	No	No, solo para versiones en desuso	BS	NTB		URL+VI+Nee +1+	Manual
The Mole	No	Yes	BS	NTB		URL+VI+Nee	Manual
V3n0m	Si	No	No	NTB, TB	No	URL	Automático
Shoryuken - SQL	No	No	No	No		URL+VI	Manual

Injection Takeover							
FJ-Injector Framework	No	No, solo para versiones en desuso	Bit a bit	NTB		URL+VI+Nee	Manual
ISR-SQLGet	No	No	No	No		Archivos Configuración	Manual
Priamos	No	No	No	No		URL	Manual
BSQLBF	No	No, solo para versiones en desuso	Bit a bit	NTB		URL+VI+Nee	Manual

Fuente: Elaboración propia

Con respecto al tiempo empleado por cada herramienta para la extracción de información, la tabla 21 resume la evaluación realizada. Todas las herramientas listadas en la tabla mencionada, extrajeron correctamente la información, aunque solo SQLMAP es compatible con los dos tipos de inyecciones SQL inferenciales: basados en tiempo y no basados en tiempo.

Tabla 21 *Tiempo de evaluación de las herramientas disponibles*

Herramienta	**Tiempo de evaluación de las herramientas al riesgo de inyecciones SQL inferenciales basadas en tiempo**	**Tiempo de evaluación de las herramientas al riesgo de inyecciones SQL inferenciales no basadas en tiempo**
Absinthe	No compatible	5 minutos
Sqlmap	2 horas 14 minutos 26 segundos	1 minuto 39 segundos
BSQL Hacker	No compatible	2 minutos 3 segundos
SQLBrute - SQL Injection Brute Forcer	1 hora 20 minutos 42 segundos	No compatible
The Mole	No compatible	1 minuto 24 segundos

Fuente: Elaboración propia

5. Propuesta de una herramienta para la evaluación de inyecciones SQL inferenciales

5.1 Análisis y diseño de la herramienta

En esta sección, se incluyen las consideraciones realizadas para el análisis de la aplicación. Este se basa en la metodología indicada.

5.1.1 Análisis de requerimientos

Con respecto a los requerimientos del sistema, se realizó este trabajo de investigación como fuente primaria de información. Por lo anterior, no se realizaron entrevistas a personas externas.

- **Requerimientos funcionales**

En la tabla 22 se encuentran los requerimientos funcionales.

Tabla 22 *Requerimientos Funcionales*

Id	Requerimiento	Categoría
R1	Identificar automáticamente las entradas del sitio web	Identificación
R2	Identificar automáticamente el tipo de base de datos	Identificación
R3	Identificar si la entrada es vulnerable a un ataque de inyecciones SQL inferenciales	Identificación
R4	Extraer el metadata (nombre base de datos, tablas y columnas) de la base de datos	Evaluación
R5	Extraer la información de la base de datos	Evaluación
R6	Mostrar tiempos empleados en la extracción	Evaluación

Fuente: Elaboración propia

Estos requerimientos fueron obtenidos a partir de la sección 3 y la sección 4.3 del presente documento.

- **Requerimientos no funcionales**

Con respecto a los requerimientos no funcionales, se determina la interfaz, seguridad, comunicaciones y el lenguaje de programación.

- **Interfaz**

Teniendo en cuenta la interfaz de las herramientas más utilizadas, la herramienta realizada será stand-alone, en modo consola de comandos.

- **Seguridad**

La herramienta solicitada no tendrá administración de usuarios y no requerirá acceso por usuario y contraseña al iniciar. Se deberá realizar validación a las entradas con el fin de evitar superar el buffer definido. No se llevará control de las auditorías realizadas ni los accesos hechos por parte de los usuarios.

- **Comunicaciones**

El software deberá comunicarse vía http con el sitio web vulnerable. La comunicación del sitio web con la herramienta será a través del código HTML producido, que no será interpretado sino analizado por el software desarrollado.

- **Portabilidad**

La herramienta desarrollada deberá funcionar en los sistemas operativos Windows y Linux.

- **Lenguaje de programación**

Para la definición del lenguaje de programación, se consultó el lenguaje de programación de las herramientas listadas en la sección anterior. Con el objetivo de determinar esta característica, se revisó la documentación existente,

los requisitos de instalación y, en algunos casos, se aplicó ingeniería inversa sobre el ejecutable. El resultado de este análisis, se muestra en la tabla 23. Usando la información de esta tabla y consultando los tiempos tomados por las herramientas, documentado en la tabla 21, se determinó que las herramientas que extrajeron la información en menos tiempo fueron las desarrolladas en el lenguaje de programación Python. Por esta razón, se escoge este lenguaje de programación para realizar la nueva herramienta.

Tabla 23 *Lenguaje de programación utilizado por las herramientas analizadas*

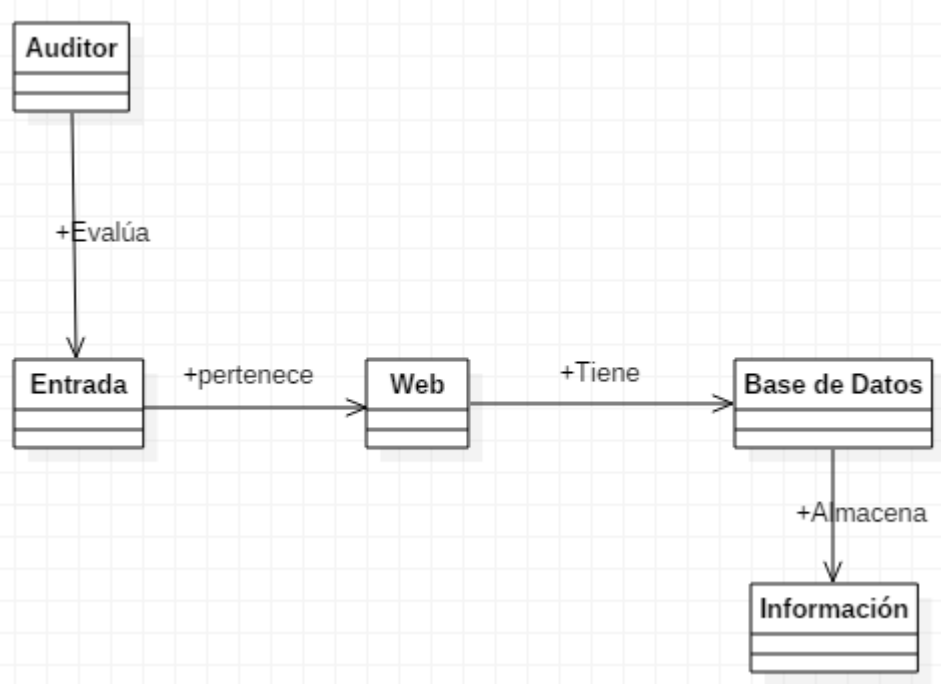
Herramienta	**Lenguaje**	**Herramienta**	**Lenguaje**	**Herramienta**	**Lenguaje**
Absinthe	.NET	SQLBrute - SQL Injection Brute Forcer	Python	SQID - SQL Injection digger	Ruby
Marathon Tool	.NET	SQLNinja	Perl	Safe3 SQL Injector	No identificable
SQLiX	PERL	BBQSQL	Python	The Mole	Python
Acunetix 11 Trial	.NET	Squeeza	Ruby	V3n0m	Python
Sqlmap	Python	Havij	.NET	Shoryuken - SQL Injection Takeover	Linux bash
Wapití	Python	SQL Power Injector	.NET	FJ-Injector Framework	C++
Paros	Java	BlindCanSeeQL	.NET	ISR-SQLGet	Perl
Pixy	Java	Pangolin	No identificable	Priamos	.NET
BSQL Hacker	.NET	DarkMysqli	Python	BSQLBF	Perl

Fuente: Elaboración propia

5.1.2 Modelado del dominio

Con el objetivo de entender el dominio del sistema de información a desarrollar, el modelo que está en la figura 15 indica los participantes en la evaluación del riesgo de inyecciones SQL inferenciales. Un auditor realiza su evaluación sobre cada entrada del sistema con el objetivo de extraer la información contenida en la base de datos.

Figura 15 *Modelado de dominio*

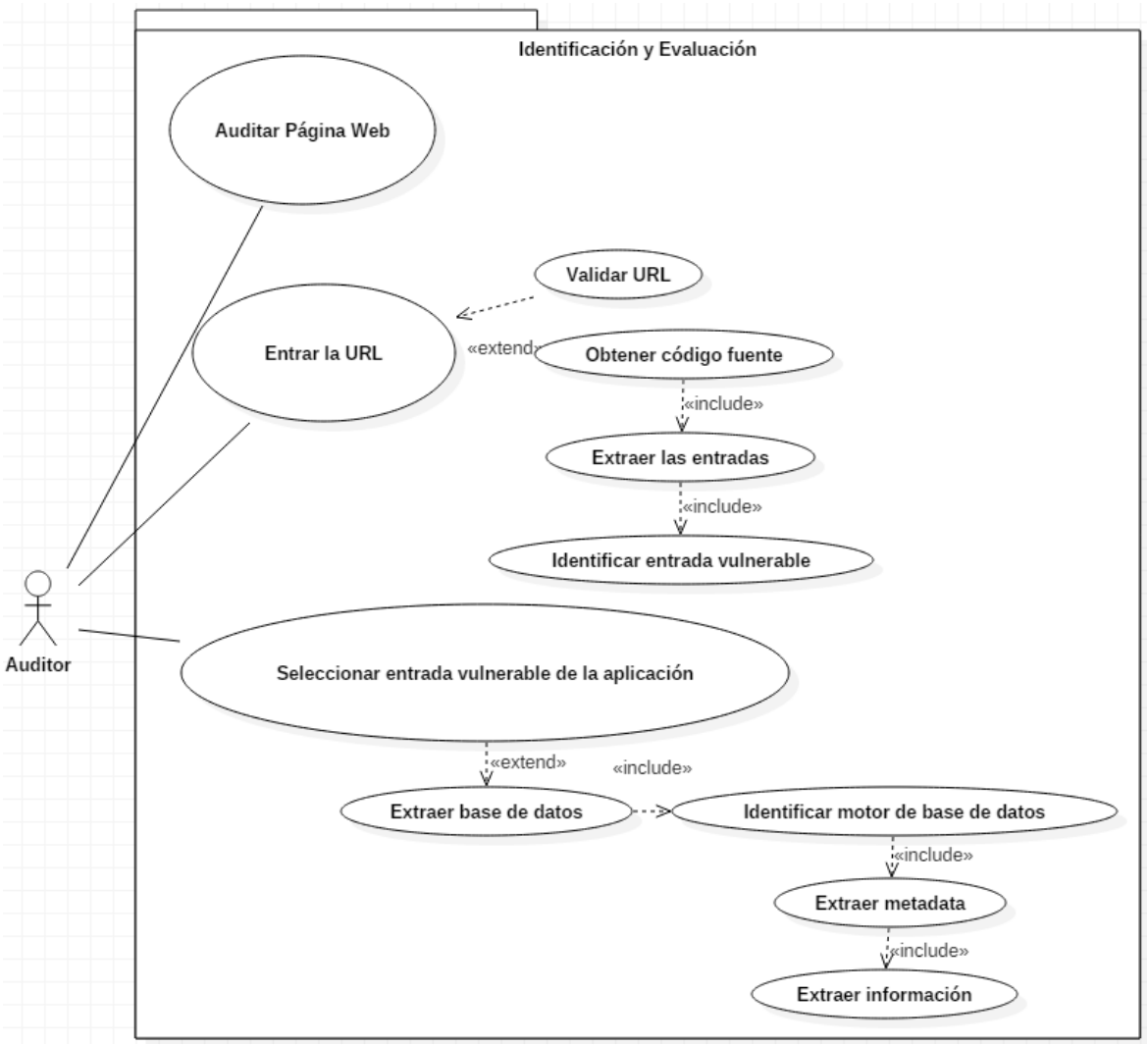


Fuente: Elaboración propia

5.1.3 Casos de uso

Teniendo en cuenta que el uso del sistema de información será la identificación y evaluación de los riesgos de inyecciones SQL, se realiza el diagrama de este caso de uso en la figura 16.

Figura 16 *Diagrama de caso de uso*



Fuente: Elaboración propia

▪ Descripción

En la figura 16, se puede ver la interacción que tendrá el usuario y las interacciones internas entre los componentes implicados en la evaluación del riesgo. En la tabla 24 está la descripción de este de manera más completa.

Tabla 24 *Descripción caso de uso*

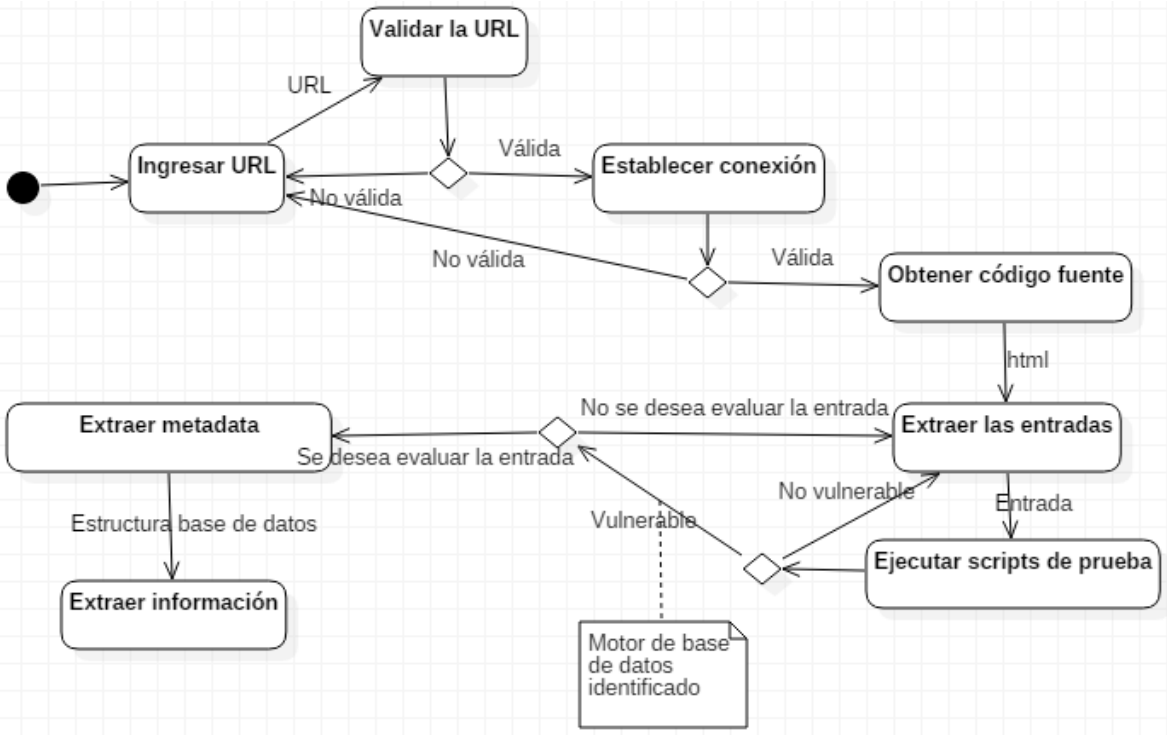
Caso de uso	1
Nombre del caso de uso	Identificación y evaluación de inyecciones SQL inferenciales
Actores	Auditor de sistemas de información web
Descripción	Un auditor ingresa el sitio web al que desea identificar y evaluar la presencia del riesgo de inyecciones SQL inferenciales
Tipo	Primario y esencial
Referencias cruzadas	Funciones R1, R2, R3, R4, R5, R6
Pre-Condición	Se cuenta con conexión a la URL ingresada por el auditor.
Post-Condición	La información de la base de datos es extraída
Escenario principal	El auditor inicia la ejecución e ingresa la URL del sitio web que desea analizar. El sistema valida la URL y si tiene conexión, descarga el código fuente de la página. A partir del código fuente, extrae las entradas de la página y realiza una batería de pruebas para saber si esa entrada es vulnerable. Esta lista de entradas vulnerables se muestra al usuario. Este elige la entrada con la cual se evaluará el riesgo. El sistema identifica el motor de la base de datos, extrae su metadata y la información, la cual es mostrada al usuario.
Escenario alternativo	El usuario decide no escoger una opción de la lista de entradas vulnerables y sale de la ejecución.

Fuente: Elaboración propia

5.1.4 Diagrama de actividades

Para establecer las actividades del único caso de uso que tiene el sistema de información, se realiza el diagrama de actividades mostrado en la figura 17.

Figura 17 *Diagrama de actividades*



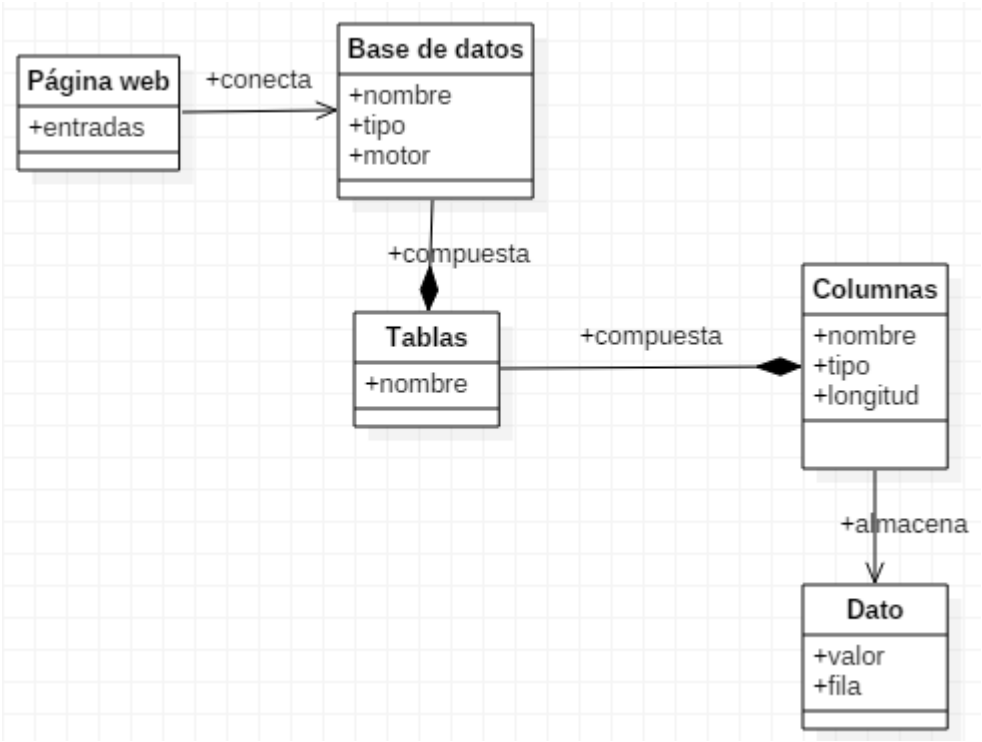
Fuente: Elaboración propia

Con base a este diagrama de actividades, se procede a realizar el diagrama de clases.

5.1.5 Diagrama de clases

El diagrama de clases contiene una descripción gráfica de la estructura interna de la herramienta. Este diagrama se encuentra en la figura 18.

Figura 18 *Diagrama de clases*

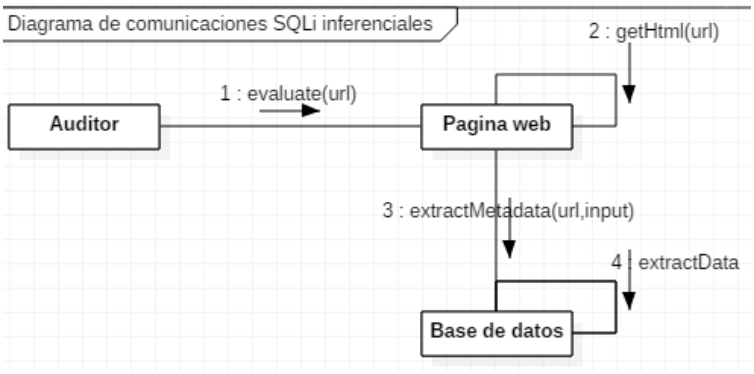


Fuente: Elaboración propia

5.1.6 Diagrama de comunicaciones

Este diagrama de comunicaciones, muestra la relación interna entre los componentes del software. Se encuentra en la figura 19.

Figura 19 *Diagrama de comunicaciones*

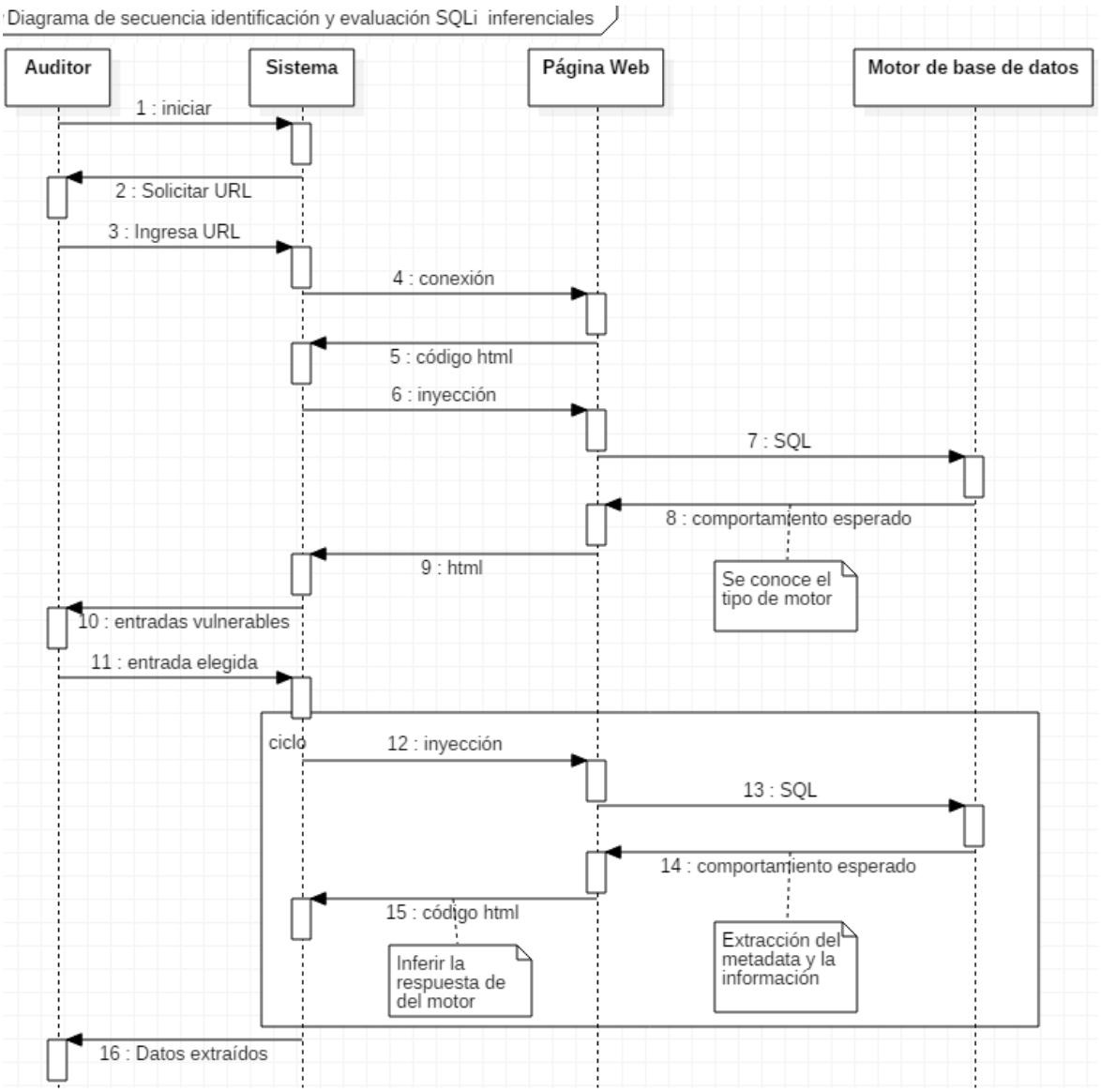


Fuente: Elaboración propia

5.1.7 Diagrama de secuencia

El diagrama de secuencia muestra la interacción interna entre los componentes y participantes de la evaluación del riesgo en cuestión. Este diagrama se muestra en la figura 20.

Figura 20 *Diagrama de secuencia*

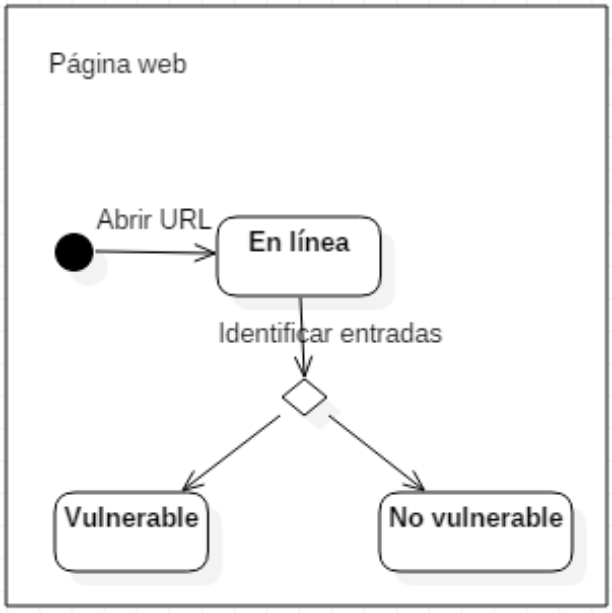


Fuente: Elaboración propia

5.1.8 Diagrama de estados

En este diagrama, solo se seleccionó el componente del sistema que puede tener estados diferentes. Este componente es la página web, la cual puede estar en línea, vulnerable o no vulnerable. Este componente se encuentra en la figura 21.

Figura 21 *Diagrama de estados*

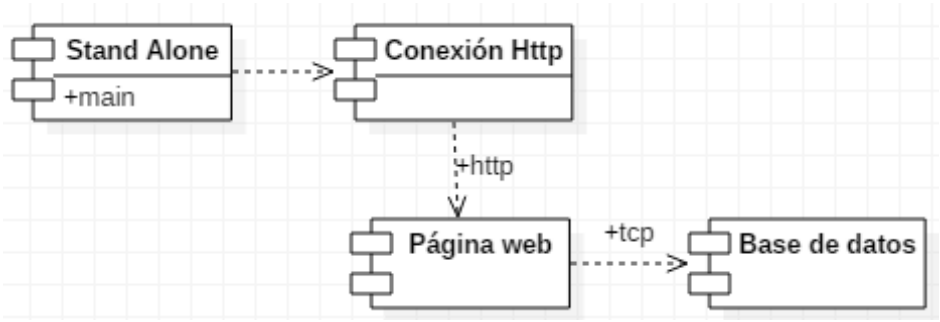


Fuente: Elaboración propia

5.1.9 Diagrama de componentes

El diagrama de componentes permite observar la conectividad de los componentes del sistema. Se encuentra en la figura 22.

Figura 22 *Diagrama de componentes*

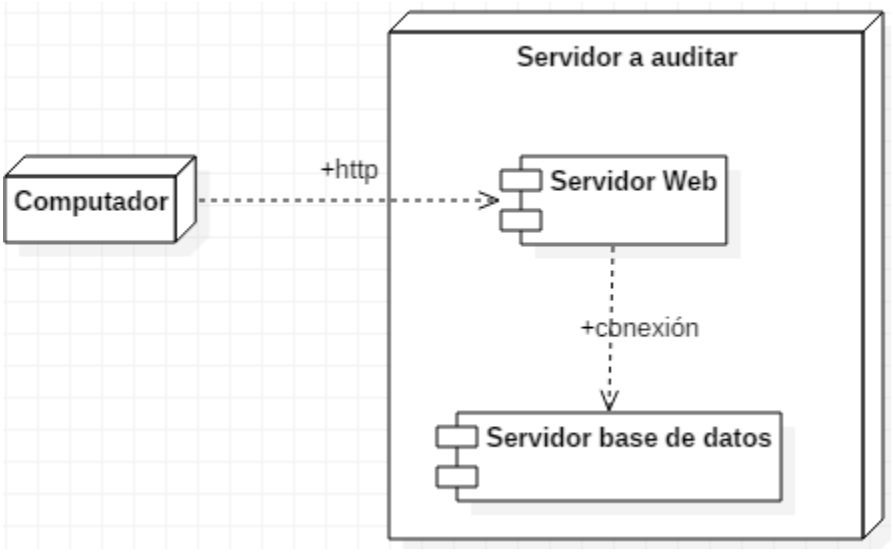


Fuente: Elaboración propia

5.1.10 Diagrama de despliegue

Este diagrama demuestra el despliegue físico del sistema de información. De acuerdo con los requerimientos establecidos, solo se requiere un computador conectado a la red para usar el sistema. Este diagrama se encuentra en la figura 23.

Figura 23 *Diagrama de despliegue*



Fuente: Elaboración propia

5.1.11 Diseño del algoritmo principal

De acuerdo con el análisis realizado en el capítulo 4, el algoritmo bit a bit puede llegar a ser el más eficiente si se cumplen con las condiciones establecidas en dicha sección.

Para esto, se determinó el número 256 como cantidad de caracteres UNICODE máxima debido a que toman todos los caracteres usados en los idiomas inglés y español. Con este parámetro, la tabla 25 muestra los criterios específicos del retraso de tiempo para la construcción de la herramienta.

Tabla 25 *Valores Δ específicos para 256 valores UNICODE*

Valores Δ	Secuencial	Binario	Bit a bit
$< 30\delta$	3	2	1
$> 30\delta$ $\lesssim 35\delta$	2	3	1
$> 30\delta$ y $\gtrsim 35\delta$	1	3	2

Fuente: Elaboración propia

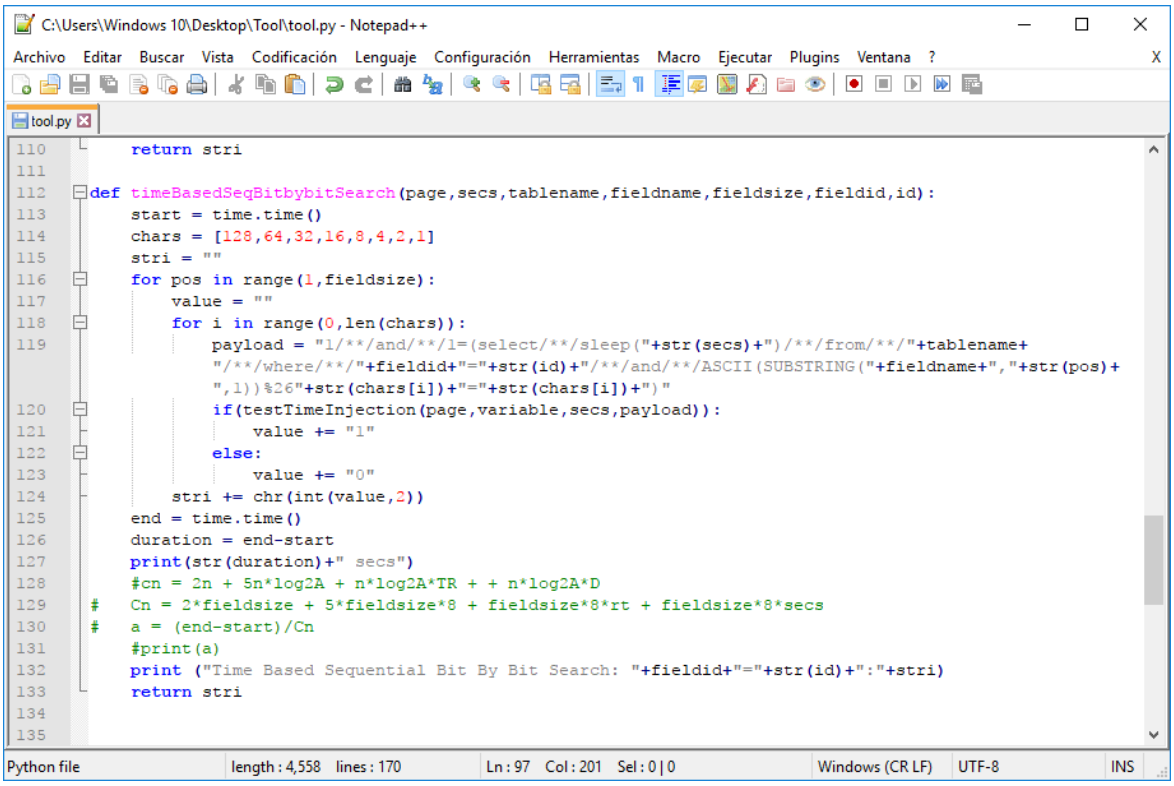
Tomando en cuenta lo anterior, se eligió el algoritmo bit a bit con un valor de retraso de tiempo de 30 veces el tiempo de carga de la página.

5.2 Implementación

5.2.1 Algoritmo principal

De acuerdo con el análisis y diseño, se procedió a implementar el algoritmo principal de extracción en la herramienta. Este algoritmo se encuentra en la figura 24.

Figura 24 *Algoritmo bit a bit implementado en la herramienta*



Fuente: Elaboración propia

5.2.2 Optimización

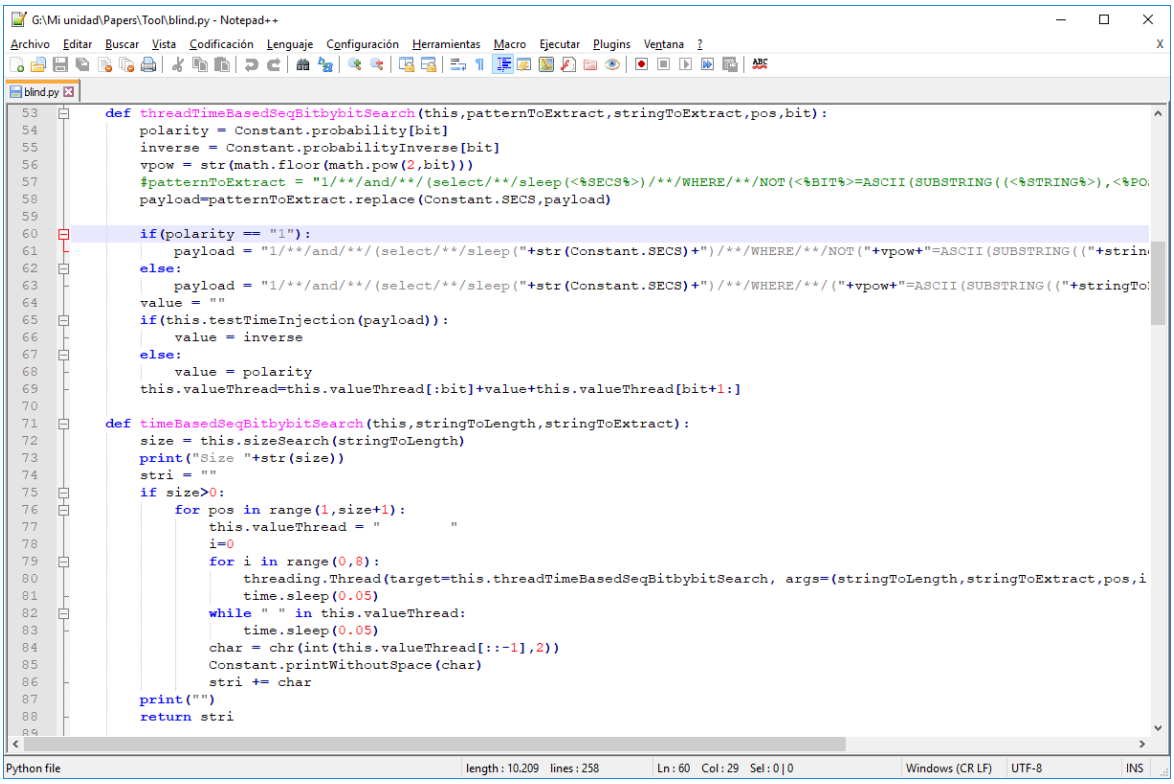
Además de implementar el algoritmo, este se optimiza usando hilos de multiprocesamiento para cada campo de la base de datos.

La estrategia desarrollada fue la siguiente:

1. Establecer la estructura interna de la base de datos.
2. Para cada bit a extraer, de manera paralela se inicia un hilo por cada registro de la tabla. Allí, se ejecuta el algoritmo bit a bit, con un valor de tiempo igual a 30 veces el tiempo de cargue normal del entorno web.
3. Se muestran los resultados, calculando el tiempo empleado.

En la figura 25 se encuentra la optimización elegida para el algoritmo.

Figura 25 *Optimización del algoritmo bit a bit en la herramienta desarrollada*

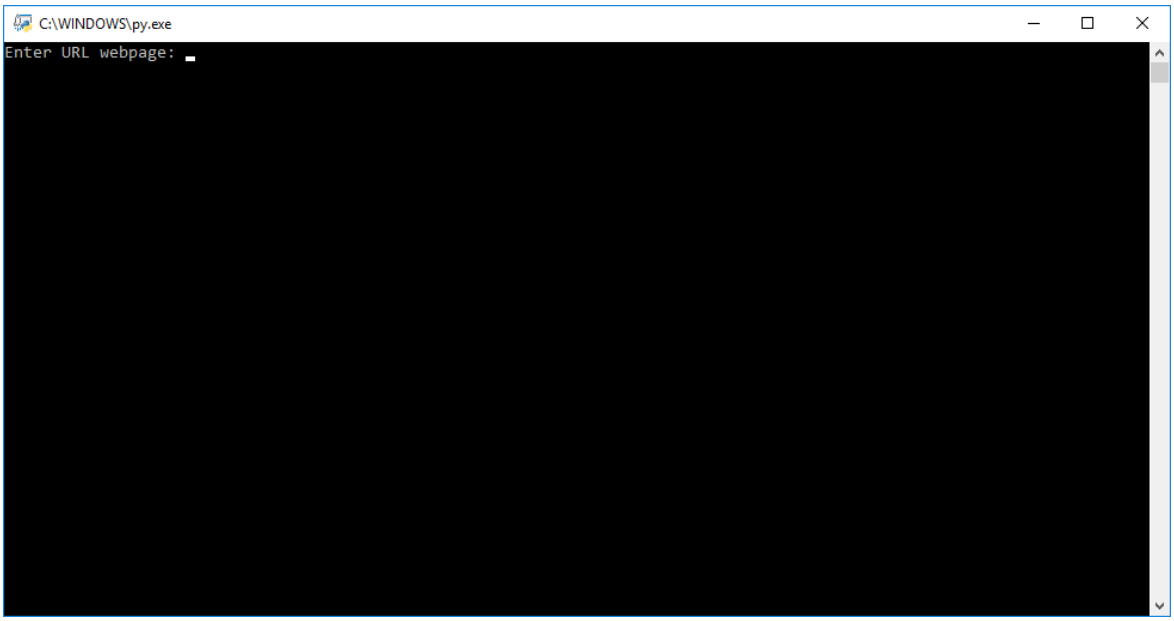


Fuente: Elaboración propia

5.2.3 Ejecución

En las figuras 26, 27 y 28 se puede observar el funcionamiento de la herramienta desarrollada. De acuerdo con el análisis realizado y la interface a la que los auditores están acostumbrados, se puede observar que la herramienta es un programa stand-alone por consola de comandos (figura 26).

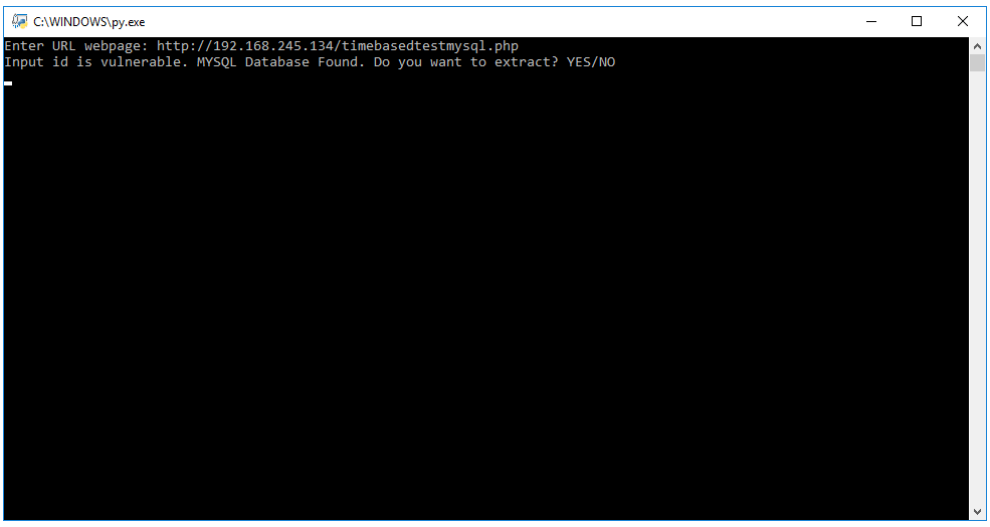
Figura 26 *Inicio aplicación desarrollada*



Fuente: Elaboración propia

Para su ejecución, esta herramienta solo solicita la URL de la página a auditar y cuando encuentra una entrada vulnerable, pide confirmación sobre si se desea extraer la información de la base de datos a través de esa entrada (figura 27).

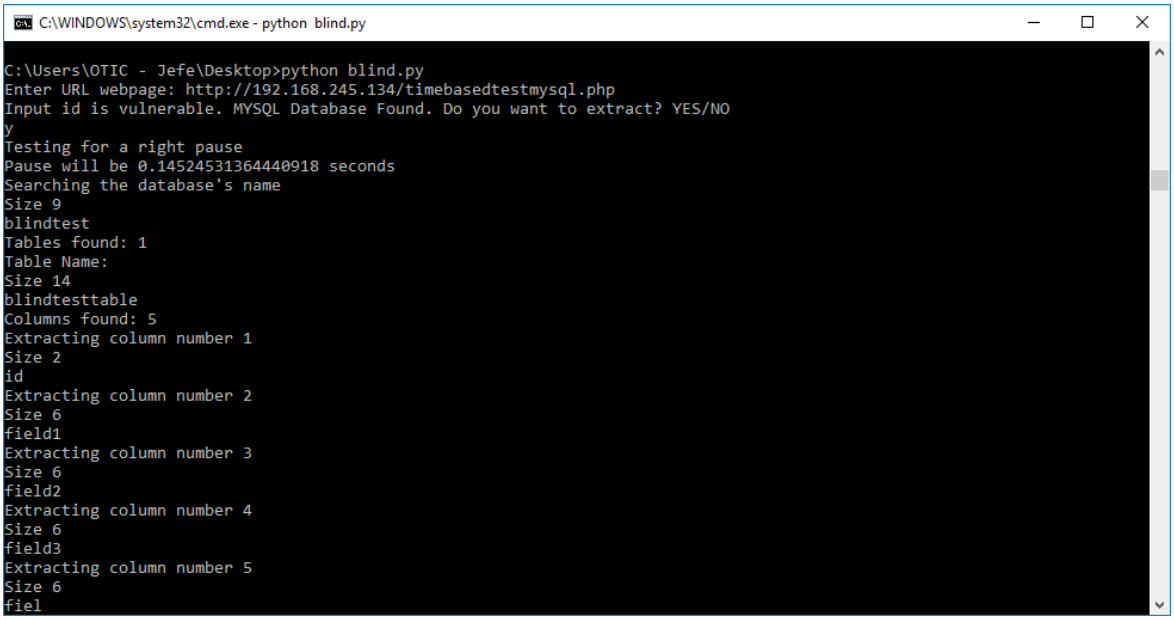
Figura 27 *Confirmación de extracción por parte del usuario*



Fuente: Elaboración propia

Cuando el auditor confirma, la herramienta procede a calcular el valor de la pausa, de acuerdo al tiempo de carga de la página. A continuación, extrae el metadata de la base de datos y la información almacenada (Figura 28).

Figura 28 *Extracción de información con la herramienta desarrollada*



Fuente: Elaboración propia

5.3 Validación

5.3.1 Validación en entorno de pruebas

Antes de realizar la validación en un entorno real, primero se realizó la misma comparación hecha sobre las herramientas existentes que se usó para determinar cuál era la más eficiente de las disponibles actualmente.

Al ejecutar la herramienta, los tiempos obtenidos fueron de 1 hora, 3 minutos y 40 segundos para el caso de las inyecciones inferenciales basadas en tiempo y de 1 minuto y 10 segundos para el caso de las no basadas en tiempo. Para facilitar la comparación, la tabla 26 muestra los tiempos de todas las herramientas analizadas y el tiempo de la herramienta desarrollada. Puede notarse que se redujo el tiempo sobre las herramientas existentes.

Tabla 26 *Tiempos de ejecución de las herramientas incluyendo la desarrollada*

Herramienta	**Tiempos de evaluación del riesgo de inyecciones SQL inferenciales basadas en tiempo**	**Tiempos de evaluación del riesgo de inyecciones SQL inferenciales NO basadas en tiempo**
Herramienta desarrollada	1 hora 03 minutos 40 segundos	1 minuto 10 segundos
Absinthe	No compatible	5 minutos
Sqlmap	2 horas 14 minutos 26 segundos	1 minuto 39 segundos
BSQL Hacker	No compatible	2 minutos 3 segundos
SQLBrute - SQL Injection Brute Forcer	1 hora 20 minutos 42 segundos	No compatible
The Mole	No compatible	1 minuto 24 segundos

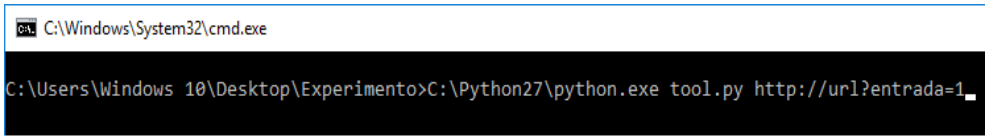
Fuente: Elaboración propia

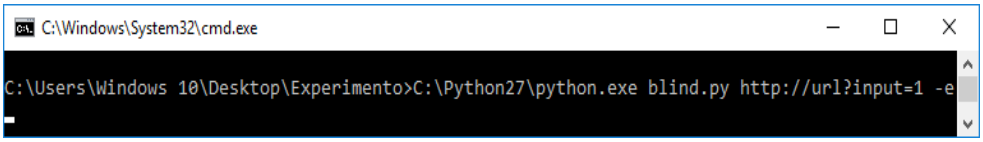
5.3.2 Validación en un entorno real

- **Planeación de la auditoría**

Para la validación de la herramienta, se usó el sistema de información académico de la Universidad Nacional de Colombia Sede Manizales. Con el fin de desarrollar la auditoría que permitió validar la herramienta, se definió el programa de auditoría mostrado en la tabla 28, basado en las recomendaciones dadas por la Asociación de Control y Auditoría de Sistemas de Información (ISACA, 2016).

Tabla 27 *Paso a paso del programa de auditoría planeado*

Paso a paso del programa de auditoría/aseguramiento
Planeación y Alcance de la Auditoría
Definir los objetivos de la auditoría/aseguramiento. Definir y revisar los objetivos de auditoria/aseguramiento
Definir los límites de la revisión La revisión debe tener definido un alcance. Entender los procesos CORE del negocio y su alineamiento con T.I.
Obtener una descripción de toda la utilidad del sistema de información a analizar.
Obtener una descripción de toda la infraestructura que soporta el sistema de información.
Identificar los usuarios del Sistema de información .
Obtener y revisar cualquier reporte previo de auditoria con planes de mitigación. Identificar observaciones abiertas y las evidencias actualizadas respecto a dichas observaciones.
Identificar y documentar el riesgo de inyecciones SQL. Realizar la evaluación del riesgo de inyecciones SQL sobre el sistema de información.
Identificar puntos de entrada del Sistema de información. Estos puntos de entrada son aquellos campos en donde un usuario puede ingresar información. También, todos los parámetros internos de la aplicación que se envían de un formulario a otro, sean visibles en la interfaz gráfica o no.
Ejecutar la herramienta desarrollada en esta tesis de maestría indicando la URL del entorno web y el punto de entrada a auditar. Así: 

Revisar la respuesta de la herramienta sobre la presencia del riesgo de inyecciones SQL.
Identificar si la organización ya ha identificado los puntos de entrada a la aplicación con la presencia del riesgo de inyecciones SQL.
Evaluar la eficacia de los planes de tratamiento dados a cada punto de entrada usando la herramienta desarrollada. 
Identificar si la valoración del riesgo dada por la organización corresponde a la criticidad identificada.
Discutir el riesgo con la administración del Sistema de información .
Realizar la documentación de la auditoría Documentar hallazgo y observaciones de la auditoría. Entregar la información resultante de la identificación y evaluación del riesgo de inyecciones SQL.

Fuente: Elaboración propia

En la figura 29, se encuentra la página principal de este sistema de información, el cual está desarrollado sobre el servidor web apache Tomcat y con base de datos ORACLE. Por seguridad de la Universidad, no se detallará más información sobre este sistema de información, ni su servidor ni su infraestructura.

Figura 29 *Sistema de Información Académico - UNAL Manizales*

Fuente: Elaboración propia

▪ Realización del programa de auditoría

Siguiendo el programa de auditoría realizado, se inició con la definición de los objetivos de la auditoría:

1.1 Objetivo de Auditoría: Realizar un diagnóstico sobre la presencia del riesgo de inyecciones SQL en el sistema de información académico de la Universidad Nacional Sede Manizales.

1.2 Alcance de la Auditoría: Esta auditoría se limita al sistema de información académico de la Universidad Nacional Sede Manizales. Igualmente, sobre dicho sistema, solo se tendrán en cuenta las entradas de la aplicación que no requieran autenticación. Las URL a analizar serán las siguientes:

http://sia.manizales.unal.edu.co y http://sia.manizales.unal.edu.co/FORE

1.2.1 Utilidad del Sistema de información: El sistema de información académico de la Universidad Nacional sede Manizales perfila a sus usuarios de acuerdo a su vinculación con la Universidad. Así, a los estudiantes, les permite consultar su historia académica, además, inscribir asignaturas, verificar sus notas y realizar transacciones de acuerdo a funciones establecidas.

1.2.2 Infraestructura del Sistema de Información: La infraestructura del Sistema de Información Académico se mantiene confidencial por razones de seguridad.

1.2.3 Usuarios del sistema de información: Los usuarios del sistema de información son los estudiantes de la Universidad Nacional Sede Manizales y sus docentes.

1.2.4 Informes previos de auditoría: Se analiza informe previo realizado por la contraloría. Todos los hallazgos que lo ameritaban, tienen planes de acción cerrados.

1.3 Se realiza la evaluación sobre la infraestructura del Sistema de Información Académico.

1.3.1 Puntos de Entrada del Sistema de Información.

URL: http://sia.manizales.unal.edu.co

<input class="campoLogin" id="nombre" name="nombre" size="15" type="text">

<input class="campoLogin" id="password" name="password" size="15" type="password">

<input class="btn" id="btn_inicio" onmouseout="this.id='btn_inicio';" onmouseover="this.id='btn_inicio2';" type="submit" value="iniciar sesión">

URL: http://sia.manizales.unal.edu.co/FORE

<input class="campoLogin" id="nombre" name="nombre" size="15" type="text">

<input class="campoLogin" id="password" name="password" size="15" type="password">

<input class="btn" id="btn_inicio" onmouseout="this.id='btn_inicio';" onmouseover="this.id='btn_inicio2';" type="submit" value="iniciar sesión">

<input value="015779202118420744272:qk5u5rpebla" type="hidden" name="cx">

<input value="FORID:11" type="hidden" name="cof">

<input class="field" size="15" name="q">

<input name="submit" type="image" value="Buscar" src="/FORE/images/buscargo.gif" alt="go" align="top">

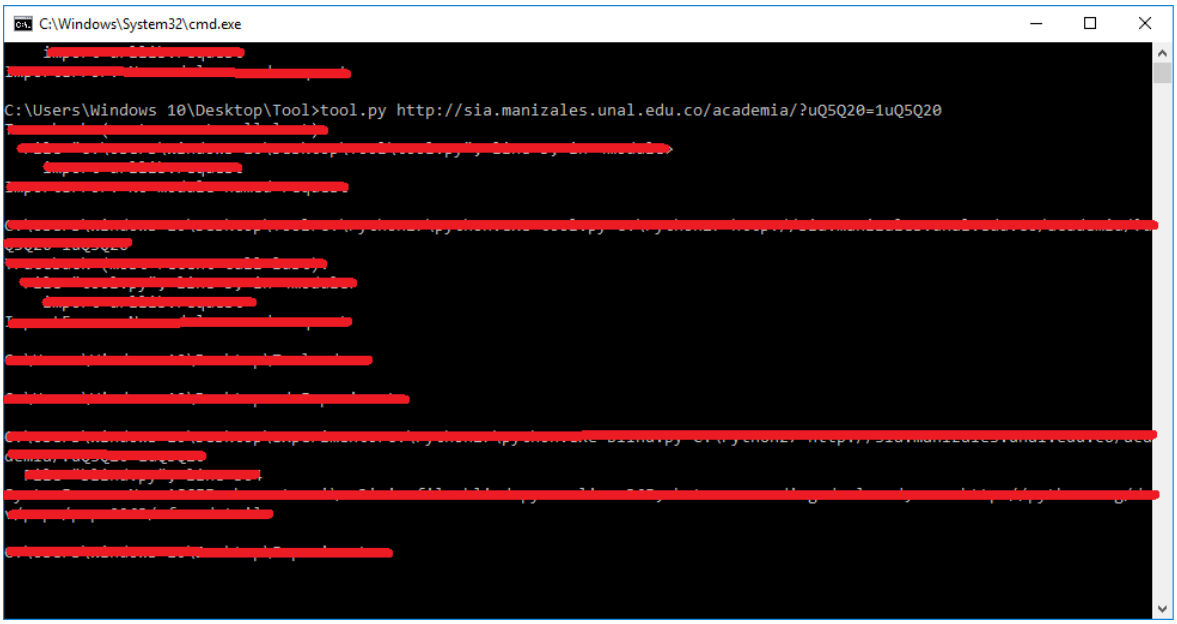
<input id="uQ5Qe" class="z-textbox" value="" type="text">

<input id="uQ5Qh" class="z-textbox" value="" type="password">

<input id="uQ5Q20" class="z-textbox" value="" type="text">

1.3.2 Para cada entrada anterior, se debe ejecutar la herramienta desarrollada en esta tesis de maestría, así como se muestra en la figura 30.

Figura 30 *Ejecución paso de auditoría 1.3.2*



Fuente: Elaboración propia

1.3.3 Para cada salida de la herramienta, se analizó el resultado mostrado para identificar la presencia del riesgo de inyecciones SQL.

Se identificó una entrada con la presencia del riesgo de inyecciones SQL.

1.3.4 Se consultó la matriz de riesgos del sistema de información, encontrando que no existe en dicha matriz ningún riesgo relativo a las inyecciones SQL.

1.3.5 No aplica. No existen planes de tratamiento.

1.3.6 No aplica, no se ha valorado el riesgo.

1.3.7 Se notifica a la Universidad sobre la presencia de este riesgo en el sistema de información académico. Según lo indicado, este riesgo desaparecerá debido a que el sitio web en cuestión se migrará a otro sistema. Toman nota sobre la ubicación de este riesgo.

1.3.8 Se entrega a la Universidad, la documentación correspondiente a este riesgo. No se anexa en este trabajo debido a la criticidad del riesgo en cuestión.

- **Resultado**

En una de estas entradas, la herramienta encontró la presencia del riesgo de inyecciones SQL. A través de esta vulnerabilidad, se extrajo la información de contenida en la base de datos ORACLE a través de las herramientas analizadas. En la tabla 28, se encuentra un resumen de los tiempos tomados en la evaluación. Es necesario mencionar información más detallada de este proceso solo se entregará a las personas encargadas dentro de la organización.

Tabla 28 *Tiempos de evaluación del entorno real analizado*

Herramienta	**Tiempos de la extracción de información sobre el riesgo de inyecciones SQL inferenciales no basadas en tiempo del entorno real analizado**
Herramienta desarrollada	4 minuto 10 segundos
Absinthe	15 minutos
Sqlmap	7 minuto 30 segundos
BSQL Hacker	5 minutos 3 segundos
SQLBrute - SQL Injection Brute Forcer	1 hora 31 minutos 24 segundos
The Mole	4 minuto 50 segundos

Fuente: Elaboración propia

6. Análisis de resultados, conclusiones y recomendaciones

6.1 Análisis de resultados

En la sección 3 del presente trabajo de investigación, se analizaron los algoritmos de extracción de acuerdo a cada tipo de inyección SQL inferencial, como parte del primer objetivo específico de este trabajo.

Las pruebas de penetración sobre las inyecciones inferenciales no basados en tiempo son entre 10 y 2 mil veces más eficientes que las basadas en tiempo. Por lo tanto, en el proceso de evaluación de las entradas vulnerables de un ambiente web es preferible realizar la extracción de información usando la vulnerabilidad no basada en tiempo. El realizar este proceso sobre las basadas en tiempo debe ser la última opción a considerar por parte del auditor.

Para modificar la consulta SQL original, pueden usarse 3 tipos de condicionales, los cuales no tienen una incidencia significativa en el tiempo de extracción y su uso es determinado por la compatibilidad que tenga con el motor de base de datos sobre el cuál funciona el ambiente web.

En la tabla 12, se encuentra un resumen de los tiempos de ejecución matemáticos de cada herramienta. Se nota que la diferencia de los tiempos entre las inyecciones basadas en tiempo y no basadas en tiempo radica en el parámetro Δ, el cuál es el retraso de tiempo inyectado. Si este se hace igual a cero (Δ=0), los tiempos de ejecución serán iguales. Sin

embargo, esta condición nunca se cumplirá debido a que el tiempo inyectado debe ser mayor al tiempo del retraso de la red y este último valor siempre será mayor a 0. Por lo tanto, matemáticamente se comprobó que la extracción de información a través de una entrada vulnerable a inyecciones SQL inferenciales no basadas en tiempo siempre será más eficiente.

Al comparar los tiempos de ejecución, se determinó que el algoritmo bit a bit es el más eficiente en todos los casos excepto cuándo se escogen valores de inyección altos, lo cual está determinado a través de una fórmula matemática resumida en la tabla 13. Esto pudo comprobarse a través del trabajo empírico realizado, durante el cual se colocaron los algoritmos con diferentes parámetros y en los cuales se comprobó lo hallado de manera teórica. Para realizar este trabajo empírico fue necesario calcular el tiempo de respuesta del ambiente web a analizar. Este valor es siempre diferente y depende de los tiempos de respuesta de la red, los tiempos del servidor web, la concurrencia y muchos otros factores que impiden definir un valor único. Se establecieron tres maneras para el cálculo de este valor. Una manera es realizar un llamado a la página web previo a cada inyección y usar el tiempo tomado en este para calcular el tiempo a inyectar. La otra manera fue realizar un número alto de llamados al sitio web y usar el peor tiempo recibido. Con estas dos maneras, el tiempo de extracción no fue eficiente. La última forma consiste en hacer un número alto de peticiones y tomar el valor promedio. Esta opción es la más óptima en términos del costo/beneficio, ya que los tiempos de extracción se mantuvieron estables.

En este trabajo, se definió el índice de precisión del algoritmo, el cual permite establecer la cota inferior del tiempo a inyectar. A valores más bajos, peor es el índice de precisión. De acuerdo con el estudio realizado, los valores óptimos para inyectar deben ser mayores a 16 veces el retraso de la red (tabla 17) con el fin de reducir falsos positivos en la extracción. Además, se establecieron 3 optimizaciones comunes: probabilidad de ocurrencia, uso de diccionarios y el uso de hilos. Estas optimizaciones pueden convivir juntas en una misma implementación, lo que permite agilizar la extracción de la información.

Como parte del primer objetivo específico, en la sección 4, se analizaron 27 herramientas de auditoría disponibles o documentadas en la literatura científica para la evaluación del riesgo estudiado y se realizó una comparación exhaustiva de cada una de ellas.

Después del análisis realizado, es notable la falta de soporte a las herramientas actuales: solo 3 de las 27 herramientas analizadas cuentan con este. La falta de soporte activo, ocasiona que las herramientas no sean compatibles con las últimas versiones de los motores bases de datos. Durante los años 2004 a 2007, se crearon el 77.7% de las herramientas analizadas. Esto se debe a que, durante esas fechas, no existía una herramienta estable para la evaluación del riesgo. Durante los últimos años, la herramienta SQLMAP es la única de esa época con soporte actual y con un periodo de madurez de más de una década. Esta herramienta se incluye en las versiones de Linux especializadas en pruebas de penetración y esto probablemente explica que las otras herramientas quedaran en desuso y sin soporte.

Con respecto a la compatibilidad de las herramientas, la base de datos más común para el desarrollo de las auditorías es el motor de base de datos Microsoft SQL Server, seguido por MySQL y Oracle. Esto aumenta la facilidad para explotar la vulnerabilidad de inyecciones SQL sobre dichos motores, haciendo más crítico la mitigación del riesgo en los ambientes web que los utilizan.

Solo cuatro herramientas analizan las entradas del sitio web identificando las que son vulnerables a una inyección SQL. Esta funcionalidad es muy útil para un auditor, ya que simplifica el trabajo de detectar la presencia de este riesgo. Si se toma el riesgo de inyecciones SQL inferenciales, solo tres (Acunetix, Wapití y V3n0m) detectaron las entradas vulnerables lo cual pone de manifiesto la dificultad que tiene este subtipo de inyecciones para identificar y mitigar. Sin embargo, de estas tres herramientas, ninguna realiza una evaluación completa del riesgo (Acunetix no lo incluye en la versión libre).

Para la extracción de información del riesgo, solo la herramienta SQLMAP tiene la opción para evaluar ambos tipos de inyecciones inferenciales: basadas en tiempo y no basadas en tiempo. Por lo tanto, esta herramienta es una buena opción cuando no se conoce específicamente el tipo de inyecciones inferenciales al cual es vulnerable el sitio web (lo

cual es frecuente). Sin embargo, al no poseer una opción de identificación, con esta herramienta es necesario ingresar de manera manual cada entrada a la aplicación.

En cuanto a la información solicitada al usuario por parte de las herramientas, las cuatro (15%) que realizan la identificación, solicitan únicamente la URL de la página web a auditar. El 44% requieren que se indique, además de la URL, la entrada de la aplicación a analizar. El resto de herramientas solicitan más información como el cambio a buscar sobre la página web para el caso de las inyecciones no basadas en tiempo y, en algunos casos, requieren que se indique manualmente los puertos de conexión y hasta archivos de configuración extensos. Teniendo en cuenta que la mayoría de estas herramientas no tienen documentación vigente, se requiere de tiempo y conocimiento en programación para entender qué parámetros se solicitan de modo que funcionen correctamente. Por tal razón, es mejor utilizar herramientas que sean fáciles de configurar, como las que solo solicitan la URL o, a lo máximo, las que solicitan también la entrada vulnerable. Entre menos parámetros requiera la aplicación, menos tiempo perdido tiene el auditor. Y con el objetivo de agilizar el proceso, 8 herramientas (29%) funcionan de manera automatizada, por lo que, después de indicar los parámetros necesarios, la herramienta inicia automáticamente la extracción. Solamente el 18,5% de las herramientas extrajo la información. El resto no era compatible con las versiones actuales de base de datos o ni siquiera generaba un mensaje de error y la opción de extraer no funcionaba.

El algoritmo más usado por las herramientas es la búsqueda binaria, seguida por la extracción bit a bit y por la búsqueda secuencial optimizada. Esto demuestra la importancia de este trabajo de investigación, ya que el algoritmo de búsqueda binaria requiere de condiciones muy específicas para ser el mejor de los algoritmos analizados. Estas condiciones no se dan en las herramientas analizadas.

La tabla 29 resume el análisis de los resultados de la comparación de las herramientas. Solo aparecen en el listado las que extrajeron correctamente la información o identificaron correctamente el riesgo. Como puede verse en esta tabla, para la identificación de las entradas vulnerables, las herramientas Acunetix o V3n0m son las más recomendadas debido a que cuentan con soporte actual. Teniendo en cuenta las condiciones de la versión demo de Acunetix, es probable que, para la mayoría de los casos, sea mejor usar v3n0m por su licenciamiento libre.

Tabla 29 *Análisis de la comparación de herramientas realizada*

Herramienta	Soporte Activo	Identificación a partir de URL	Extracción de inyecciones basadas en tiempo y no basadas en tiempo	Máximo 2 parámetros para su funcionamiento	Herramienta Automatizada	Algoritmo bit a bit
Absinthe	No	No	No	Si	No	No
Acunetix Trial	Si	Si	No incluido	Si	Si	No
SQLMAP	Si	No	Si	Si	Si	Si
Wapití	No	Si	No	Si	SI	No
BSQL Hacker	No	No	No	Si	Si	No
SQLBrute - SQL Injection Brute Forcer	No	No	No	Si	No	No
The Mole	No	No	No	No	No	No
V3nom	Si	Si	No	No	Si	No

Fuente: Elaboración propia

Con respecto a la extracción, la herramienta SQLMap es la más completa teniendo en cuenta que extrae la información de las inyecciones SQL inferenciales basadas en tiempo y no basadas en tiempo, cuenta con soporte activo, no solicita más de dos parámetros para su funcionamiento, es una herramienta automatizada e implementa el algoritmo más

rápido. Sin embargo, para el caso de las inyecciones basadas en tiempo, la herramienta SQLBrute – SQL Injection Brute Forcer es la más rápida. Esto se debe a que implementó dos optimizaciones al algoritmo de búsqueda secuencial: hilos y probabilidad de ocurrencia.

Como parte del segundo objetivo de este trabajo de investigación, se implementó una herramienta de auditoría usando el algoritmo bit a bit optimizado a través de hilos. Este algoritmo puede paralelizarse de manera completa: un hilo por cada campo de la base de datos, otro por cada letra del campo y uno por cada bit. Sin embargo, al implementar estos hilos, la herramienta empezó a realizar tantas peticiones al servidor web, que terminó realizando una denegación de servicios. Por tal razón, la herramienta solo cuenta con hilos para extraer cada bit. Esta optimización fue clave para mejorar la velocidad de la extracción e incidió directamente en que se pudiera superar los tiempos de la herramienta SQLBrute, la cual, como ya se mencionó, cuenta con dos optimizaciones al algoritmo de búsqueda secuencial.

La herramienta desarrollada puede ser utilizada por auditores con o sin conocimiento de las inyecciones SQL inferenciales, esto porque, posterior a la validación efectuada en esta investigación, implementó la funcionalidad de identificación a partir de la URL y la extracción automática a partir de las entradas vulnerables. Es necesario realizar mayores pruebas sobre la funcionalidad de identificación antes de liberar una versión para uso público.

Finalmente, como parte del proceso de validación de la herramienta y respondiendo al tercer objetivo específico, se validó en un entorno real. Esto generó la necesidad de hacer una herramienta compatible con más bases de datos, ya que la página en la que se probó está realizada sobre el motor Oracle.

En el proceso de identificación, se encontró una entrada vulnerable a una inyección SQL. Aunque no era una inyección inferencial, se forzó a las aplicaciones a extraer la información como si fuera una vulnerabilidad inferencial no basada en tiempo. En esta validación, la herramienta desarrollada presentó un mejor tiempo en la evaluación.

Además, se demostró que, a pesar de ser el principal riesgo de seguridad en un entorno web, su presencia sigue siendo común en entornos corporativos.

6.2 Conclusiones

Sobre el análisis de los algoritmos, técnicas y herramientas, se establecieron las siguientes conclusiones.

1. Uno de los mayores aportes de esta investigación es que no solo describe, explica y analiza los algoritmos encontrados, sino que determina las condiciones favorables para el uso de cada uno. Así se estableció que el algoritmo bit a bit es el mejor algoritmo analizado para la extracción de la información.
2. Varios autores determinan de manera aleatoria el tiempo para la inyección, por ejemplo, (Anley, 2002b; W. G. J. Halfond et al., 2008; Štampar, 2016) usan 5 segundos, (Chandrasekhar & Singh, 2014) usa 10 segundos. Sin embargo, esta investigación ha comprobado la relación entre el tiempo de carga de la página web y el retardo inyectado, definiendo fórmulas matemáticas para su cálculo que distan de los autores antes citados. La determinación del tiempo a inyectar no es un valor aleatorio.
3. Para la implementación y aplicación de las formulas definidas, se debe calcular el tiempo de carga de la página web como el promedio del tiempo de carga de un número alto de peticiones al sitio antes de iniciar la extracción.
4. Para la definición y comparación de los algoritmos de extracción, es necesario tener en cuenta el Índice de Precisión del Algoritmo (IPA) definido en este trabajo de investigación.
5. Hasta antes de la construcción de la herramienta asociada a esta investigación, la mejor herramienta para una evaluación completa del riesgo era SQLMAP. Sin embargo, hablando sobre velocidad de extracción, la herramienta SQLBrute - SQL Injection Brute Forcer extrajo la información en el menor tiempo para el caso de las inyecciones inferenciales basadas en tiempo y The Mole para el caso de las no basadas en tiempo. Debe tenerse en cuenta que ambas herramientas funcionan de manera manual por lo que el conocimiento del auditor sobre el riesgo debe ser mayor.

Con respecto al desarrollo de la herramienta, se puede concluir que:

1. La herramienta desarrollada es la mejor alternativa para la extracción de la información. No solo realiza de manera más eficiente el riesgo, sino que la función de identificación agiliza la aplicación de las pruebas.
2. PYTHON ha sido el lenguaje de programación en el cual se han implementado las herramientas más eficientes.
3. Las optimizaciones a los algoritmos son claves para el desarrollo de una herramienta eficiente, aunque debe tener cuidado con el uso de hilos y, en general, con toda aquella optimización que pueda generar una denegación de servicios.

Después de validar la herramienta diseñada en la evaluación del riesgo inyecciones SQL inferenciales en el entorno web del sistema de información académico de la Universidad Nacional de Colombia se puede ver que:

1. La presencia del riesgo de inyecciones SQL en dicho entorno confirma las observaciones realizadas por la organización OWASP sobre la alta probabilidad de la presencia del riesgo.
2. Es posible usar el programa de auditoría propuesto para auditar el riesgo de inyecciones SQL en un entorno web.
3. La herramienta desarrollada reduce los tiempos de evaluación en un 13% aproximadamente con respecto a la herramienta The Mole en un entorno real y en un 20% con respecto a la herramienta SQLBrute - SQL Injection Brute Forcer en un entorno de pruebas.

6.3 Trabajo futuro y recomendaciones

Existen 7 diferentes tipos de inyecciones SQL. Esta tesis de maestría analizó el tipo llamado inferenciales. Por lo tanto, existen otros 6 tipos de inyecciones SQL que deben analizarse a profundidad, con el objetivo de abarcar el riesgo de manera completa.

Además, es necesario realizar un análisis sobre las metodologías de pruebas de penetración y análisis de vulnerabilidad con el objetivo de determinar la que mejor aplica al entorno colombiano.

Los desarrolladores de ambientes web, deben aplicar con cuidado las recomendaciones de código seguro y las técnicas de autores que han investigado la prevención de inyecciones SQL (Atoum & Qaralleh, 2014; Bandhakavi, 2007; Chandrashekhar, Mardithaya, Thilagam, & Saha, 2012b; W. Halfond & Orso, 2007; Sajjadi & Tajalli Pour, 2013; Shar & Tan, 2013; Tajpour, Ibrahim, & Masrom, 2011).

6.4 Logros: principales contribuciones

Estas son las principales contribuciones de esta investigación:

- Se determinó cuál es el mejor algoritmo para la implementación de herramientas para la identificación del riesgo de inyecciones SQL inferenciales.
- Se realizó una herramienta más eficiente para esta identificación.
- Se propuso un índice de precisión de los algoritmos para inyecciones SQL inferenciales.

6.5 Difusión de los resultados

Se presentó el artículo “A comparison of binary and sequential search algorithms to evaluate the risk of inferential SQL injections in web applications” a la revista Revista IEEE América Latina con ISSN 1548-0992 y homologada por Colciencias como A2. Al momento de escribir esta tesis, este artículo se encuentra en proceso de revisión.

Durante la semana del 26 de septiembre de 2016, se realizó en la ciudad de Popayán el Congreso Colombiano de Computación organizado por la Sociedad Colombiana de Computación, por la Fundación Universitaria de Popayán, la Universidad del Cauca y por IEEE Colombia. En este evento, se presentó una ponencia titulada: “Conditions of Binary

search algorithm for being an optimization of sequential inference algorithms to audit the risk of SQL injections in web environments”

7. Bibliografía

Aliero, M. S., Ardo, A. A., & Ghani, I. (2016). Classification of Sql Injection Detection And Prevention Measure, *6*(2), 6–17.

Alonso, C., Daniel Kachakil, Bordón, R., Guzmán, A., & Beltrán, M. (2007). Time-Based Blind SQL Injection using Heavy Queries A practical approach for MS SQL and Marathon Tool. Retrieved from https://technet.microsoft.com/en-us/library/cc512676.aspx

Anley, C. (2002a). Advanced SQL injection in SQL server applications. *White Paper, Next Generation Security Software …*. Retrieved from http://alsouza.googlecode.com/svn/trunk/Monografia/subsidios/sqlinjection/Advanced SQL Injection.pdf%5Cnhttps://sparrow.ece.cmu.edu/group/731-s11/readings/anley-sql-inj.pdf

Anley, C. (2002b). More advanced SQL injection. Retrieved from http://www.cgisecurity.com/lib/more_advanced_sql_injection.pdf

Atoum, J. O., & Qaralleh, A. J. (2014). A HYBRID TECHNIQUE FOR SQL INJECTION ATTACKS DETECTION AND PREVENTION. *International Journal of Database Management Systems (IJDMS)*, *6*(1). https://doi.org/10.5121/ijdms.2014.6102

Bandhakavi, S. (2007). CANDID : Preventing SQL Injection Attacks using Dynamic Candidate Evaluations.

Bandhakavi, S., Bisht, P., Madhusudan, P., & Venkatakrishnan, V. N. (2007). CANDID: Preventing Sql Injection Attacks Using Dynamic Candidate Evaluations. *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 12–24. https://doi.org/10.1145/1315245.1315249

Benedikt, M. (1992). *Cyberspace: Some Proposals*. *Cyberspace: First Steps*.

Berners-Lee, T. J. (1992). The world-wide web. *Computer Networks and ISDN Systems*, *25*(4–5), 454–459. https://doi.org/10.1016/0169-7552(92)90039-S

Bsides. (2016). BSides Co. Retrieved December 9, 2016, from https://www.bsidesco.org//

BSIDES. (2016). BSides. Retrieved December 9, 2016, from www.securitybsides.com/

Cameron Hotchkies. (2004). Automating Blind SQL Exploitation.

Cerrudo, C. (2002). Manipulating Microsoft SQL Server Using SQL Injection. *Application Security, Inc*, 1–14. Retrieved from http://america.securimetric.net/library/software/Manipulating_SQL_Server_Using_SQL_Injection.pdf%5Cnhttp://www.blackhat.com/presentations/win-usa-03/bh-win-03-cerrudo/bu-win-03-cerrudo-notes

Chandrasekhar, U., & Singh, D. (2014). Understanding Query Vulnerabilities for Various SQL Injection Techniques, *243*, 1063–1075. https://doi.org/10.1007/978-81-322-1665-0

Chandrashekhar, R., Mardithaya, M., Thilagam, S., & Saha, D. (2012a). SQL Injection Attack Mechanisms and Prevention Techniques. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 7135 LNCS, pp. 524–533). https://doi.org/10.1007/978-3-642-29280-4_61

Chandrashekhar, R., Mardithaya, M., Thilagam, S., & Saha, D. (2012b). SQL Injection Attack Mechanisms and Prevention Techniques. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 7135 LNCS, pp. 524–533). https://doi.org/10.1007/978-3-642-29280-4_61

Chenyu, M., & Fan, G. (2016). Defending SQL Injection Attacks based-on Intention-Oriented Detection, (Iccse), 939–944.

Clarke, J., & Alvarez, R. M. (2009). *SQL Injection Attacks and Defense*. *SQL Injection*

Attacks and Defense (Vol. 54). https://doi.org/10.1016/B978-1-59-749963-7.00001-3

Craigen, D., Diakun-Thibault, N., & Purse, R. (2014). Defining cyber-security. *Technology Innovation Management Review*, (October), 13–21.

DEF CON. (2016). Def Con. Retrieved December 9, 2016, from https://www.defcon.org/

DragonJar. (2016). DragonJar Con. Retrieved December 9, 2016, from https://www.dragonjarcon.org/

Du, W. (1998). Vulnerability Testing of Software System Using Fault Injection, 1–20.

Evteev, D. (2010). Methods of Quick Exploitation of Blind Sql Injection. *http://www.ptsecurity.com/download/PT-Devteev-FAST-Blind-SQL-Injection.pdf*.

Faker, S. A., Muslim, M. A., & Dachlan, H. S. (2017). 24.A Systematic Literature Review on SQL Injection Attacks Techniques and Common Exploited Vulnerabilities. *International Journal of Computer Engineering and Information Technology*, *9*(12), 284–291.

Farmer, D., & Venema, W. (1993). Improving the Security of Your Site by Breaking Into it. *USENET Posting*.

Florescu, D., Levy, A., & Mendelzon, A. (1998). Database techniques for the World-Wide Web. *ACM SIGMOD Record*, *27*, 59–74. https://doi.org/10.1145/290593.290605

Forristal, J. (1998). NT Web Technology Vulnerabilities. *Phrack Magazine*, 8. Retrieved from http://phrack.org/issues/54/8.html

G. Stoneburner, Goguen, a., & Feringa, a. (2002). Risk Management Guide for Information Technology Systems. *National Institute of Standards and Technology, Special Publication 800 -30*, *800–30*, 55. https://doi.org/10.1111/j.1745-6622.2008.00202.x

George, T. K., & Jacob, P. (2016). A Proposed Architecture for Query Anomaly Detection and Prevention against SQL Injection Attacks, *137*(7), 11–14.

Ghafarian, A. (2017). A Hybrid Method for Detection and Prevention of SQL Injection

Attacks, (July), 833–838. https://doi.org/10.1109/SAI.2017.8252192

Gollmann, D. (1999). *Computer security*. (I. John Wiley & Sons, Ed.). New York. Retrieved from http://dl.acm.org/citation.cfm?id=292465

Halde, J. (2008). SQL Injection analysis , Detection and Prevention.

Halfond, W. G. J., Viegas, J., & Orso, A. (2008). A Classification of SQL Injection Attacks and Countermeasures. *Preventing Sql Code Injection By Combining Static and Runtime Analysis*, 53.

Halfond, W., & Orso, A. (2007). Detection and Prevention of SQL Injection Attacks. *Malware Detection*, *27*(4), 85–109. Retrieved from http://dx.doi.org/10.1007/978-0-387-44599-1_5

Hernández, R., Fernández, C., & Baptista, P. (2014). *Metodología de la investigación*. *Journal of Chemical Information and Modeling* (Vol. 53). https://doi.org/10.1017/CBO9781107415324.004

Hudák, P. (2016). Automated SQL injection attacks validation system.

ISACA. (2008). IS Auditing Guideline - G3 - USE OF COMPUTER-ASSISTED AUDIT TECHNIQUES (CAATs), (January), 3–8.

ISACA. (2016). Information Systems Auditing : Tools and Techniques. Creating Audit Programs.

ISO/IEC. (2000). ISO/IEC 15445:2000(E) Information technology — Document description and processing languages — HyperText Markup Language (HTML) Contents.

ISO/IEC. (2004). ISO 13335-1:2004 Information technology -- Security techniques -- Management of information and communications technology security -- Part 1: Concepts and models for information and communications technology security management.

ISO/IEC. (2007). ISO/IEC 19011:2012 Directrices para la auditoría de los sistemas de

gestión, (571).

ISO/IEC. (2009). NTC ISO 27005:2009 Tecnología de la información. Técnicas de seguridad. Gestión del riesgo en la seguridad de la información, (571).

ISO/IEC. (2011). ISO/IEC 9075-1:2011 Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework).

ISO/IEC. (2012). ISO/IEC 27032:2012 Information technology — Security techniques — Guidelines for cybersecurity, *2012*.

ISO/IEC. (2013). ISO/IEC 27001:2013 Tecnología de la información. Técnicas de seguridad. Sistemas de gestión de la seguridad de la información. Requisitos, (571).

ISO/IEC. (2015). ISO/IEC 27002:2015 Tecnología de la información. Técnicas de seguridad. Código de práctica para controles de seguridad de la información.

ITU. (2015). Definition of cybersecurity, *1205*(December). Retrieved from http://www.itu.int/en/ITU-T/studygroups/com17/Pages/cybersecurity.aspx

Jansen, W., Walsh, J., Dolan, K. V., Wright, P. A., & Montequin, R. R. (2000). U.S. Department of Defense Application-Level Firewall Protection Profile for Medium Robustness Environments. Retrieved from http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA395046

Krsul, I. V. (1998). Software Vulnerability Analysis.

Lawal, M. ., Sultan, M. A. B., & Shakiru, A. O. (2016). Systemic Literature Review on SQL Injection Attacks. *International Journal of Soft Computing*, *11*(1), 26–35.

Ley estatutaria 1266. (2008). Ley estatutaria 1266 del 31 de diciembre de 2008, 1–24.

Ley estatutaria 1581 de 2012. Congreso de la república de Colombia (2012). https://doi.org/Research Study 292

Litchfield, D. (2005). Data-mining with SQL Injection and Inference. *NGSSoftware Insight*

Security Research Publication, (September).

Maor, O., & Shulman, A. (2003). Blindfolded SQL Injection. Retrieved from http://www.imperva.com/docs/Blindfolded_SQL_Injection.pdf

Nagpal, B., Singh, N., Chauhan, N., & Panesar, A. (2015). Tool based implementation of SQL injection for penetration testing. *International Conference on Computing, Communication & Automation*, 746–749. https://doi.org/10.1109/CCAA.2015.7148509

Naik, N., & Jenkins, P. (2016). Web protocols and challenges of Web latency in the Web of Things. *International Conference on Ubiquitous and Future Networks, ICUFN*, *2016–Augus*, 845–850. https://doi.org/10.1109/ICUFN.2016.7537156

Nayak, A., Poriya, A., & Poojary, D. (2013). Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems*, *5*(4), 16–19.

NIA. (2002a). NIA 15 Auditoría en un Ambiente de Sistemas de Información por Computadora. Sección 401.

NIA. (2002b). NIA 16 Técnicas de Auditoría con Ayuda de Computadora. Sección 1009.

OWASP. (2013). OWASP Top 10 - 2013. *OWASP Top 10*, 22. https://doi.org/1

OWASP. (2016). OWASP. Retrieved December 9, 2016, from www.owasp.org

OWASP. (2017). OWASP Top 10 - The Ten Most Critical Web Application Security Risks. *Owasp*, 22. Retrieved from https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf%0Ahttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:OWASP+Top+10+-2010#1

Pfleeger, C. P., Lawrence, S., & Theofanos, M. F. (1989). A Methodology For Penetration Testing, *8*, 613–620.

Quatrini, S., & Rondini, M. (2011). Blind Sql Injection with Regular Expressions Attack.

Retrieved from https://www.exploit-db.com/docs/17397.pdf

Redwood, O. (2016). Extracting Multiple Bits Per Request From Full-blind SQL Injection Vulnerabilities. Retrieved November 8, 2016, from http://howto.hackallthethings.com/2016/07/extracting-multiple-bits-per-request.html

Robert Sedgewick, & Philippe Flajolet. (1996). *An Introduction to the Analysis of Algorithms*.

Ross, K., Moh, M., & Yao, J. (2018). Multi-Source Data Analysis and Evaluation of Machine Learning Techniques for SQL Injection Detection, 1–8.

Sajjadi, S. M. S., & Tajalli Pour, B. (2013). Study of SQL Injection Attacks and Countermeasures. *International Journal of Computer and Communication Engineering*, *2*(5), 539–542. https://doi.org/10.7763/IJCCE.2013.V2.244

Shah, S., & Mehtre, B. M. (2015). An overview of vulnerability assessment and penetration testing techniques. *Journal of Computer Virology and Hacking Techniques*, *11*(1), 27–49. https://doi.org/10.1007/s11416-014-0231-x

Shar, L. K., & Tan, H. B. K. (2013). Defeating SQL injection. *Computer*, *46*(3), 69–77. https://doi.org/10.1109/MC.2012.283

Slaviero, M. (2012). Blind SQL Injection Exploitation. In *SQL Injection Attacks and Defense: Second Edition* (pp. 233–288). Elsevier. https://doi.org/10.1016/B978-1-59-749963-7.00005-0

Štampar, M. (2016). Inferential SQL injection attacks. *International Journal of Network Security*, *18*(2), 316–325.

Stasinopoulos, A., & Ntantogian, C. (2018). Commix: automating evaluation and exploitation of command injection vulnerabilities in Web applications, 1–54. https://doi.org/https://doi.org/10.1007/s10207-018-0399-z

Tajpour, A., Ibrahim, S., & Masrom, M. (2011). SQL Injection Detection and Prevention Techniques. *International Journal of Advancements in Computing Technology*, *3*(7), 82–91. https://doi.org/10.4156/ijact.vol3.issue7.11

The Object Management Group. (2005). *Unified Modeling Language : Infrastructure*. *The Object Management Group*. https://doi.org/10.1007/s00500-003-0307-x

Tian, W., Yang, J. F., Xu, J., & Si, G. N. (2012). Attack model based penetration test for SQL injection vulnerability. *Proceedings - International Computer Software and Applications Conference*, 589–594. https://doi.org/10.1109/COMPSACW.2012.108

Toews, B., & Scott Behrens. (2012). BBQSQL.

Valencia Duque, F. J., & Tamayo Arias, J. (2016). *Técnicas y herramientas de auditoría asistidas por computador*. (Universidad Nacional de Colombia, Ed.) (Primera Ed).

von Solms, R., & van Niekerk, J. (2013). From information security to cyber security. *Computers & Security*, *38*(October 2013), 97–102. https://doi.org/10.1016/j.cose.2013.04.004

Weissman, C. (1985). Penetration Testing.

Wheeler, R. (2015). BlindCanSeeQL : Improved Blind SQL Injection For DB Schema Discovery Using A Predictive Dictionary From Web Scraped Word Based Lists, (October).

Whitman, M. E., & Mattord, H. J. (2012). *Principles of Information Security*. *Course Technology*. https://doi.org/10.1016/B978-0-12-381972-7.00002-6