Smart Technologies Academic Press

Journal of Cyber Security and Risk Auditing

https://www.jcsra.thestap.com/

Check for updates

Comprehensive Study of SQL Injection Attacks Mitigation Methods and Future Directions

Mohammed Al-olaqi[1], Ahmed Al-gailani[1] and M M Hafizur Rahman[1]

[1] *Department of Computer Networks and Communications, College of Computer Sciences and Information Technology, King Faisal University, Al-Ahsa, 31982, Saudi Arabia*

ARTICLE INFO

Article History

Received: 17-05-2025
Revised: 30-07-2025
Accepted: 03-09-2025
Published: 09-09-2025

Vol.2025, No.4

DOI:

***Corresponding author. Email:** mhrahman@kfu.edu.sa

Orcid: https://orcid.org/0000-0001-6808-3373

ABSTRACT

Structured Query Language Injection Attack (SQLIA) as a form of cyber threats are among the most dangerous, easily penetrating the databases, and most web based applications. These are input validation vulnerabilities that can be used to exploit such things as Structured Query Language (SQL) commands that can be used to gain exposure to and access to privileged data, and can be leveraged for compromise of the system as a whole. With this study, we present a comprehensive as well as systematic review of traditional and modern approaches for SQLIAs detection, their mitigation and prevention. The first line of protection against such advanced threats is conventional defenses such as input validation, parameterized queries, secure error handling, but they typically fail in the presence of second order, time based, or obfuscated SQLIAs. For addressing these emerging attack vectors, researchers have developed dynamic ways in the form of pattern matching approach, anomaly detection, cryptographic techniques and artificial intelligence (AI) based security systems. It studies the rise of the use of ML and DL models, especially of Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNN), and ensemble classifiers in achieving high accuracy at detecting sophisticated SQLIAs. Though detection rates are promising, suitable use of an AI based system faces challenges of computational burden, large required datasets and lack of model explainability. The study also calls for urgent attention to emerging platforms NoSQL databases and Natural Language Interfaces to Databases (NLIDBs). Finally, this study goes deeper into the implementation and utility of proactive developer training, security development practices, as well as real time monitoring frameworks including Intrusion Detection Systems (IDS) and honeypots in augmentation of application resilience. Overall, the study suggest a multi layered, adaptive defense strategy, consisting of the real time threat detection through AI technology, behaviour assessment based on context, using federated learning over several domains. This state of the art study synthesizes existing methodologies and offers foundation for future research in cybersecurity professionals and researchers aiming to booster web apps against SQL injection vulnerabilities.

Keywords: Structured Query Language Injection Attack (SQLIA), Convolutional Neural Networks (CNNs), Intrusion Detection Systems (IDS).

How to cite the article

1. Introduction

Security of web applications reaches essential status during this modern period when essential business systems depend on them. Technological dependency has heightened the frequency along with complexity of cyber-attacks using SQL injection (SQLI) vulnerabilities. SQLI attacks exploit user input fields through database query manipulation to inject harmful SQL code which grants attackers data access and modification and deletion capabilities starting from databases. These security threats create significant harm to data reliability while destroying privacy protection along with financial security standards which requires strong defensive systems. Preventing unauthorized data access with security breaches requires full comprehension of SQLI attack methods and their implementation of defensive measures Aliero et al [1] and Nair et al [2].

SQLI attacks remain widespread based on statistical findings which reveal that web vulnerabilities stem mainly from these attacks. The research indicates that SQL injection vulnerabilities form a leading 30% subset of web application security breaches reported to authorities Nasereddin et al [3]. Development teams along with organizations need to take immediate action against these threats by improving their coding practices and implementing security frameworks because of this important data. The failure to establish preventive security solutions results in both the endangerment of confidential customer information and negative consequences for company reputation and trust levels Chowdhury et al [4] and Lawal et al [5].

A depiction of the SQL injection attack process appears in Figure 1. An attacker conducts SQL injection attacks by adding string-type input into web applications to modify and alter the SQL statements. The exploitation of this SQL vulnerability threatens to harm the database through multiple possible harmful outcomes including unauthorized database manipulation and the retrieval of confidential data. System-level commands available through this attack method can lead to service denial by the system. The SQL injection attack helps attackers go beyond authentication protocols to receive full access to remote servers. Complete access to the remote server's database. Web application data storage occurs in SQL databases with an overwhelming majority of applications running indeterminate SQL databases which operate automatically in the background. The database command in SQL syntax can be modified using syntax elements which match the user-provided data comparable to standard programming constructs. Users remote to the system can send application data through a web interface that will be processed as commands against the database.

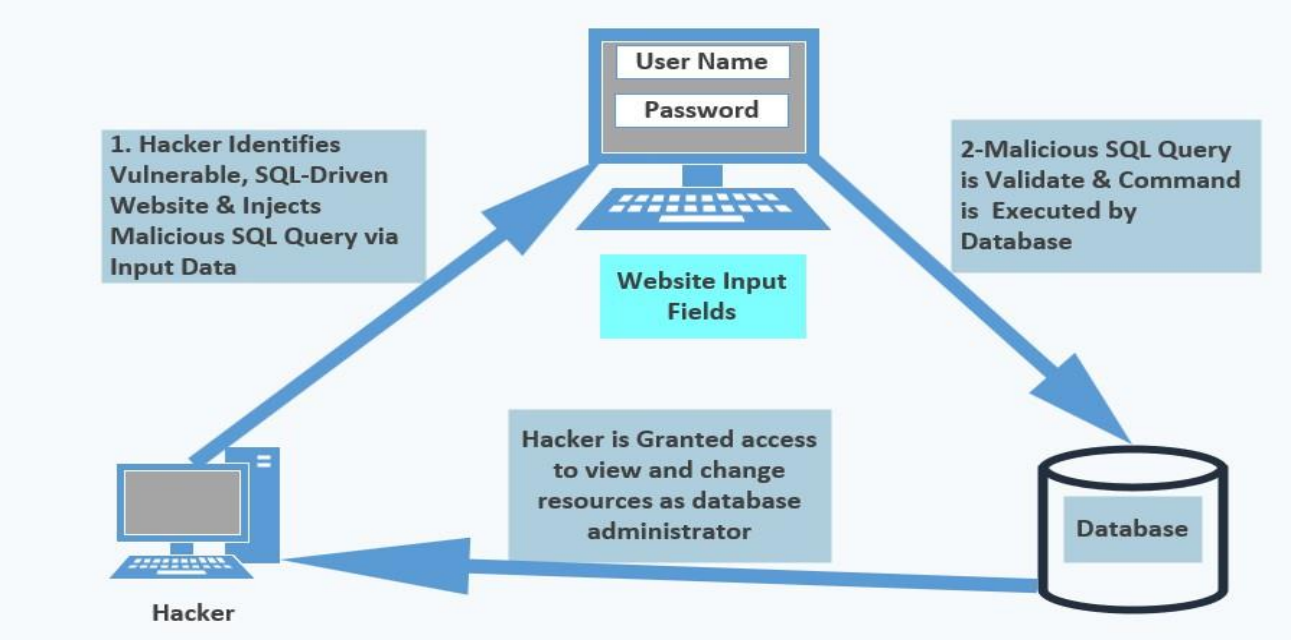


Figure 1. Illustration of the SQL Injection Attack Process in a Web Application Workflow.

The core solution to defeat SQLI vulnerabilities starts with developing applications through complete security measures across each development step. Three critical approaches to defend against SQLI vulnerabilities involve input validation and parameterized queries and efficient error handling. Input validation procedures protect the application by ensuring that only valid formats of data reach its processing functions to minimize malicious code threats. The combination of SQL commands with user inputs through parameterized queries creates a security measure which renders code injection attacks ineffective. The appropriate handling of errors works to reduce the amount of information attackers can access Hlaing et al [6].

New defensive methods against SQLI attacks have been developed in recent times, but attackers maintain their ability to discover fresh ways through traditional security measures. SQLI prevention techniques face changing threats because automated tools now let malicious actors access SQLI methods easily Alghawazi et al [7]. This evolving threat necessitates an adaptive and robust defensive posture on the part of organizations. Through integrating signature-based and heuristic detection methods security systems become more effective at discovering abnormal activity that suggests SQL injection attacks so that threats can be dealt with promptly Elshazly et al [8]. Organizations should perform regular security evaluations featuring penetration tests and vulnerability evaluations to preserve the protection of web applications against SQLI attacks. Organizations can both find security weaknesses and measure their existing defense systems through these assessment methods. By adhering to security frameworks like the OWASP Top Ten organizations can determine high-risk vulnerabilities in their applications during SQL injection prevention efforts Ali et al [9]. The need for persistent funding in security assessment procedures becomes evident because systematic testing allows organizations to detect exploitable weaknesses.

Secure coding practices require developer training to achieve the highest possible level of importance. Secure coding education and up-to-date SQLI technique training for developers produces an organization-wide security focus. Through regular training events developers can improve their skill in understanding security vulnerabilities and acquire knowledge about potential security measures Gasiba et al [10]. As developers complete their work, static and dynamic code analysis tools help them identify vulnerabilities which will be resolved before system release. SQL injection attacks continue to threaten software systems thus emphasizing the need for organizations to use multiple security layers in their applications. A robust defense strategy requires prevention mechanisms alongside assessment protocols and developer security training execution for organizations to develop strong resistant capabilities. Enhancing understanding about SQL injection vulnerabilities together with improved prevention methods will make it possible for organizations to effectively defend sensitive data from malicious attacks. Prospective security practices need adaptation to confront the permanent security threats that characterize SQLI attacks as the digital environment continues its evolution.

2. Background

2.1 SQLIA Definition

All the most of the web based applications are built on 3 tier architecture which is presented by the data tier to store structured information, business tier for processing all logical operations and presentation tier for processing user interface how the data should be presented. The above means that all three tiers are used to execute the SQLIA in SQLIA attack. Malicious SQL commands are transmitted to the business tier by exploiting the presentation layer and made to manipulate existing SQL queries. The manipulation is directed at this tier of data, in hopes of obtaining non authorized access to the information (data). The absence of proper validation in both the presentation and business tiers of web applications facilitates the success of SQL injection attacks aliero et al [10].

2.2 SQLIA Procedure

In most cases the SQL injection attack (SQLIA) procedure starts from the web interface that forwards the entered inputs to the backend server. An attacker can identify the possibility of a SQL injection by finding out from the input fields, by identifying weak validation mechanisms. It is said that the (SQLIA) is commonly used to gain access to a system bypassing the security measures. This is accomplished by embedding malicious SQL queries into user's input of the login, and processing all of them so that a true value is always yielded. In this process, Instead of writing the query from scratch every time, we modified existing SQL queries by appending the malicious input I received from the web interface the same way every time (HTTP requests). Running the newly made malicious SQL statement on the database server aliero et al [10]. Figure 2 presents the Step-by-Step Procedure of a SQL Injection Attack from Input to Execution.

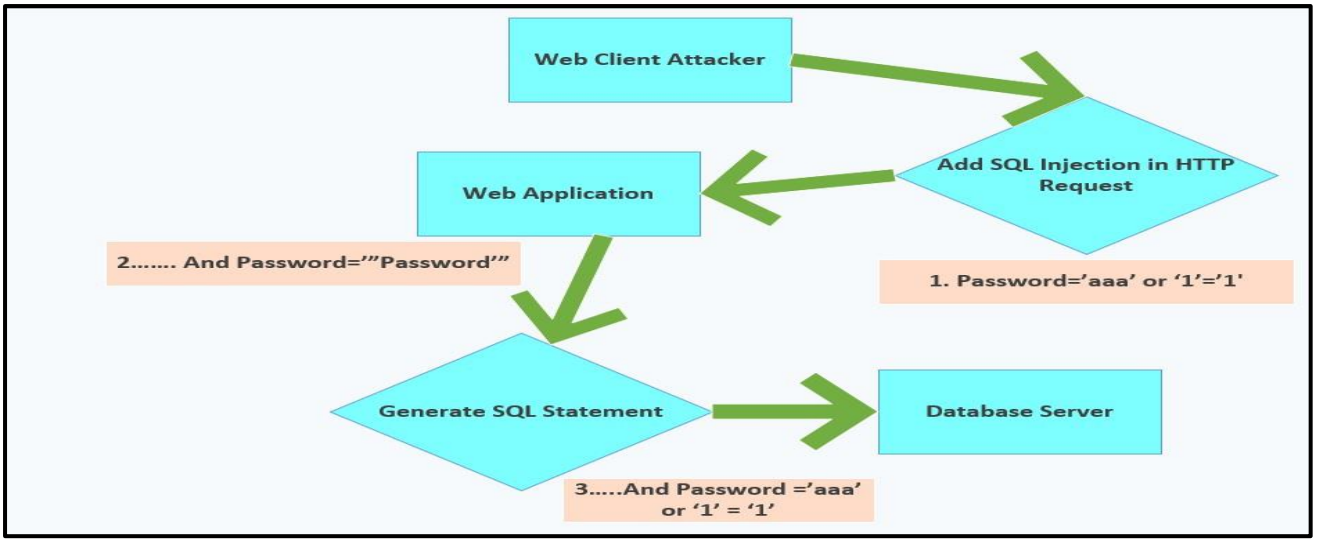


Figure 2. Step-by-Step Procedure of a SQL Injection Attack from Input to Execution.

2.3 SQLIA Consequences

Depending on the impact of SQL Injection attacks (SQLIA), SQLIA can result in disclo-sure of data and even remote control of the database with potentially many different effects. Elevation of authorization privileges become a one big outcome, as it gives the advantage to attackers to manipulate critical data using SQLIA based modifacations. Additionally, SQLIA can break the authentication mechanisms and even gain access to the system if the authentication has authenticated using SQLIA but their credentials are not right or if the validation of these credentials on the login page is badly run. For example, this breach of security could lead to loss of confidentiality since the sensitive information like credit card numbers or personal details can be revealed. SQLIA also has the ability to change, delete or lose this confidential data thus degrading its integrity.

2.4 SQLIA Attack Types

SQLIA may have different shapes. This section will show how those types of SQLI's are performed, their use, effect, and then the consequences, as shown in the Figure 3.

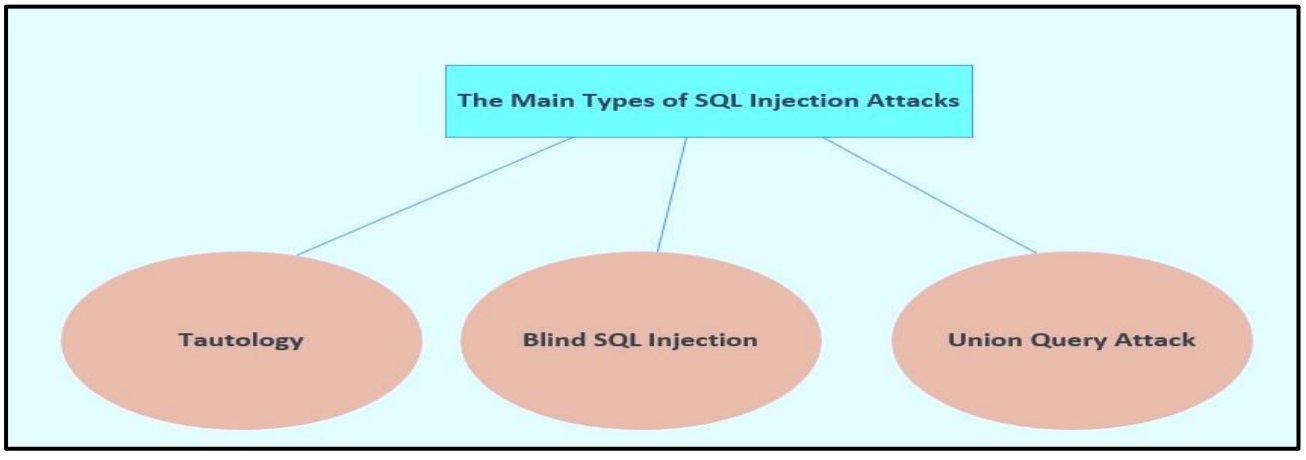


Figure 3. Classification of Common SQL Injection Attack Types.

2.4.1 Tautology

Tautology taint is an injectable field taken advantage of by an attacker to insert a malicious value that will resolve to capitalize on any condition that evaluates true. Typically, such an attack will modify the WHERE clause of a SQL statement. In the above situation, for example, an attacker can exploit this technique to bypass authentication process where the attacker modulates the SQL query to get a boolean true for any given username and password pair.

2.4.2 Blind SQL Injection

The principle of identifying actions from known untrue or known true questions put to a database which maps well to known answers. This type of SQL injection is very common when the error handling of the code used under the hood is poor, and the application will simply return a generic error. Therefore, the attacker can discover information about the database's structure and data but cannot see the outputs of the injection.

2.4.3 Union Query Attack

In this case, when an intruder appends a malicious query to the original query using UNION SQL Operator, this attack is known as union query attack. With this method, the attacker combines the results of the malicious query with the normal query, leading to an access of the SQL tables on top of the normal ones. This technique can be used by an attacker to extract data from other tables present in the database which is usually inaccessible.

2.5 SQLI Attack Goals

Hackers implement SQL Injection (SQLI) attacks for multiple functions. The main purposes of SQLI attacks consist of as shown in the Figure 4.

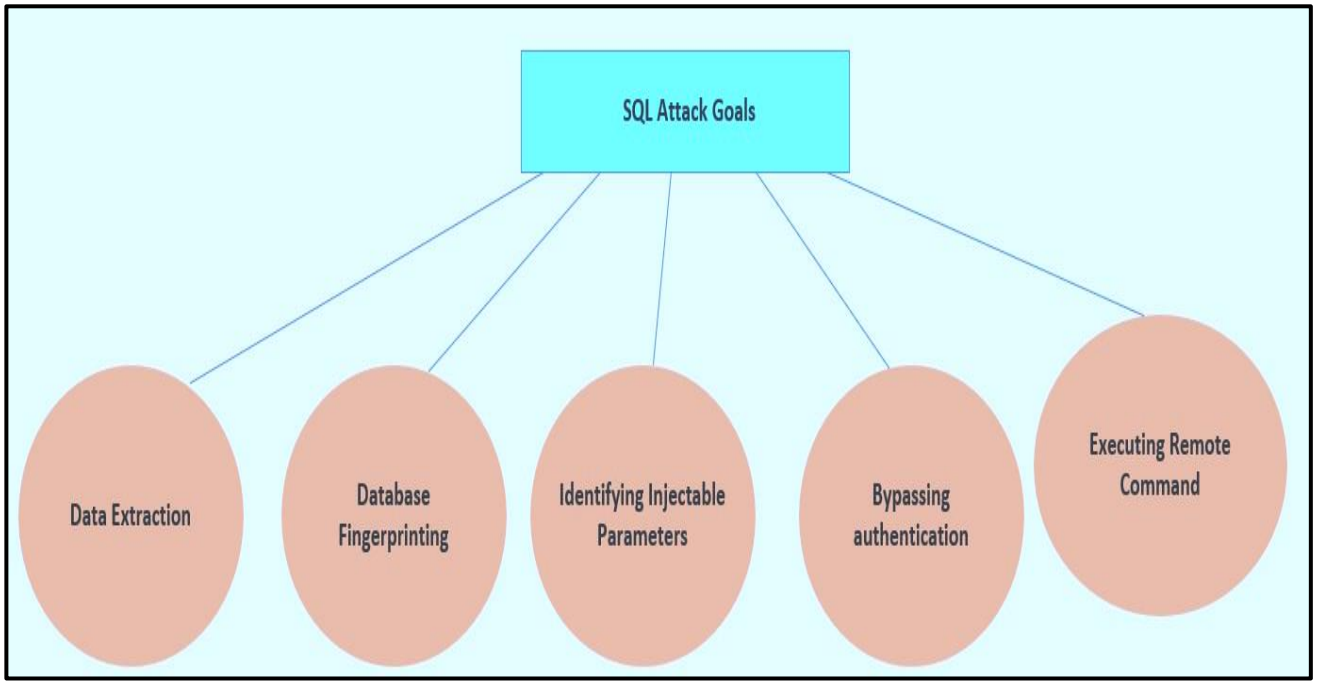


Figure 4. Strategic Objectives of SQL Injection Attacks.

2.5.1 Data Extraction

The goal of data extraction attacks involves different methods to retrieve database information. The extracted data exposes significant risks to web applications because this sensitive material is both private and confidential. The common goal pursued by SQL Injection Attack (SQLIA) attacks makes this threat the major security concern for protecting databases.

2.5.2 Database fingerprinting

To create queries suitable for a target database engine the attacker must complete database fingerprinting. The fingerprint includes all unique features which distinguish specific versions of particular database systems. Attackers need to identify the database platform type and version to create threatening SQL input since different databases like Oracle SQL Server utilize PL/SQL and Microsoft SQL Server uses T-SQL. The database reveals built-in security weaknesses to attackers due to its default features.

2.5.3 Identifying injectable parameters

Hackers first determine which fields accept input since these will become their targets for malicious code insertion. The system contains several vulnerable parameters including stored card numbers in cookies and username text boxes found in forms. An attacker can modify the statement logic through SQL code embedded within database parameters to manipulate data execution. Verifying SQL injection vulnerability requires only a single quotation mark injection because this character marks the beginning or end of strings within the SQL programming language. A related application error happens after the insertion of input which leads to vulnerability disclosure.

2.5.4 Bypassing authentication

Attackers use bypassing authentication to avoid the security systems that protect web application authentication. A successful bypass attack lets attackers access the privileges associated with another user including those of high authority.

2.5.5 Executing remote commands

The executable code present on a compromised database server is known as remote commands. Functions and stored procedures accessible to users appear among commands executed through the database. Attackers execute arbitrary database instructions through these attacks resulting in shutdown commands and disrupted database services that create a denial of service.

2.5.6 Privilege escalation

The ability to perform privilege escalation functions by exploiting both implementation flaws and logical database vulnerabilities elevates attacker privileges. The objective of these attacks targets existing database user privileges as the foundation for their infiltration instead of attempting to bypass authentication protocols. The attacker becomes root privileged after a successful exploitation attempt which results in disastrous outcomes.

3. Research Methodology

This research used a complete methodology under the PRISMA framework as illustrated in the Figure 5. The PRISMA framework conducts identification screening eligibility and inclusion tasks as four main stages. Initial paper selection for the Identification phase included 125 research papers obtained from database searches and the Saudi Digital Library (SDL). Fifty seven papers were excluded in the Screening phase because they contained duplicate information. We screened all remaining papers throughout the Eligibility phase and excluded 40 using the mentioned criteria as shown on the diagram. We concluded the Inclusion phase with a total of 28 research papers properly suited for analysis.

4. Research Contributions for SQL Injection Defense

In this research, SQL Injection (SQLi) attacks are researched comprehensively and focusedly as one of the major and severe forms of the web application vulnerabilities. This explains the basics of SQLi and how the attackers try to manipulate the backend database using unsanitized user inputs to gain unauthorized access to the sensitive data. The paper describes in details the importance of such breaches, for example data leakage, unauthorized data manipulation, and possible system compromise. The research also presents a set of secure coding principles based defense mechanisms to counter these threats. The relevance of parameterized queries and prepared statements is showcased in order to avoid malicious inputs from running SQL commands. It also provides best information validation schemes, as well as for user input covering, so the inputs were in agreement for expected formats and limitation. The contribution goes beyond basic defenses and involves application of advanced security mechanisms like use of the web application firewalls (WAFs), database activity monitoring, and intrusion detection/prevention systems (IDS/IPS) to detect and block the suspicious SQL behaviors in real time. In addition, it introduces the necessity of performing regular security audits, code study, and automated scanning for SQLi vulnerability as proactive measures to recognize and mend SQLI vulnerabilities.

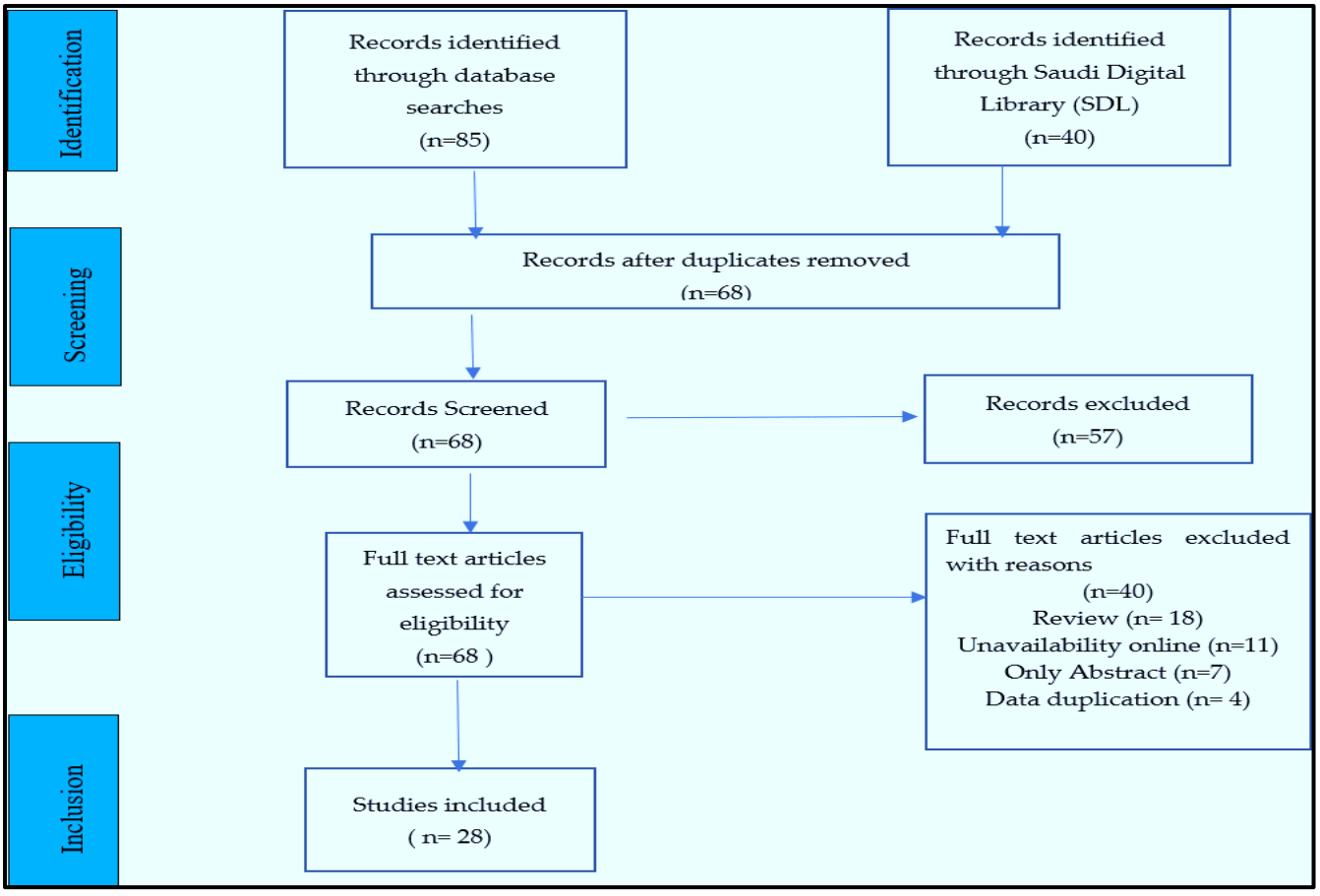


Figure 5. PRISMA Flow Diagram of Literature Selection Process for Systematic Review.

Table 1. Selected Research Papers and Their Categories in SQL Injection Defense.

Study	Title	Category
Omotunde et al [11]	A comprehensive study of security measures in database systems: Assessing authentication, access control, and beyond.	General Overview
Chowdhury et al [4]	A comprehensive survey for detection and prevention of SQL injection	Countermeasures
Hirani et al [12]	A Deep Learning Approach for Detection of SQL Injection Attacks using Convolutional Neural Networks.	Countermeasures
Hajar et al [13]	A Study of Penetration Testing Process For Sql Injection Attack	Challenges
Raut et al [14]	A study on methods for prevention of SQL injection attack	Countermeasures
Shachi et al [15]	A study on detection and prevention of SQL and NoSQL injection attack on server-side applications.	Countermeasures
Kakisim et al [16]	A deep learning approach based on multi-view consensus for SQL injection detection.	Countermeasures
Madhvan et al [17]	An Overview of Malware Injection Attacks: Techniques, Impacts, and Countermeasures.	General Overview
Augustine et al [18]	Application of Artificial Intelligence in Detecting SQL Injection Attacks.	Countermeasures

Recio et al [19]	Case-Based Explanation of Classification Models for the Detection of SQL Injection Attacks.	Challenges
Mustapha et al [20]	Comprehensive study of machine learning models for sql injection detection in e-commerce	General Overview
Wang et al [21]	Detecting command injection attacks in web applications based on novel deep learning methods.	Countermeasures
Alwan et al [22]	Detection and prevention of SQL injection attack: a study	Challenges
Alghawazi et al [7]	Detection of sql injection attack using machine learning techniques: a systematic literature review	Challenges
Shahbaz et al [23]	Evaluating CNN Effectiveness in SQL Injection Attack Detection.	Countermeasures
Odeh et al [24]	Ensemble learning techniques against structured query language injection attacks.	Countermeasures
Yunus et al [25]	Study of SQL injection: problems and prevention	General Overview
Nair and Sunil [2]	Securing Against Advanced Cyber Threats: A Comprehensive Guide to Phishing, XSS, and SQL Injection Defense	Countermeasures
Zhang et al [26]	Trojansql: Sql injection against natural language interface to database Challenges	Challenges
Paul et al [27]	SQL injection attack: Detection, prioritization & prevention Countermeasures	Countermeasures
Abdullayev et al [28]	SQL injection attack: Quick view	General Overview
Alenezi et al [29]	SQL Injection Attacks Countermeasures Assessments	Countermeasures
Kareem et al [30]	SQL injection attacks prevention system technology	Countermeasures
Mane et al [31]	SQL injection authentication security threat	Challenges
Kapoor et al [32]	SQL-Injection Threat Analysis and Evaluation	Challenges
Johny et al [33]	SQL Injection prevention in web application: a study	Countermeasures
Cahyadi et al [34]	Enhancing SQL Injection Attack Prevention: A Framework for Detection, Secure Development, and Intelligent Techniques.	Countermeasures

5. Related Works

SQL injection attacks are still one of the major security vulnerabilities that attacks web applications, databases, or enterprise systems. There have been many researchers who have in depth studied a lot of options for both detection and prevention like traditional rules based on advanced artificial intelligence and deep learning models. Researchers at Hirani et al [12] evaluated CNNs against Naïve Bayes, Decision Trees and SVMs for SQLi detection. The superior accuracy of CNNs stemmed from their automatic learning capacity regarding SQL query patterns. The high cost of large datasets coupled with significant computational expenses remains a drawback of their operation. The research team suggested that organizations should enhance the functionality of CNNs and implement deep learning approaches into cybersecurity infrastructure. The research by Raut et al [14] analyzed SQLi countermeasures which consisted of AhoCorasick pattern matching, firewalls, and IDS. Development of creative SQL injection patterns defeats pattern matching detection because pattern matching works only against known signatures. AI-based detection should work together with proven techniques to build stronger cybersecurity systems according to their recommendation. The main obstacles to detection include false positives and needed data requirements. The researchers suggested implementing a defence combination.

Kakisim et al [16] proposed a multi-view deep learning model combining Bi-LSTM and CNN for SQLi detection. Their system achieved 99.96% accuracy and outperformed conventional models. It captured sequential and structural features of SQL queries effectively. Challenges included computational cost and overfitting. They suggested hybrid models and architectural improvements for practical deployment. Augustine et al [18] integrates AI and ML and use it to increase the detection of SQLi by more than 80%. Traditional detection methods were not viable to the changes in the Evolving Attack Techniques. Moreover, organizations must adopt the AI based security frameworks. At the moment, systems lack AI integration to be as effective as they could be. It was advocating for the development of the cybersecurity evolution by the

side of the AI. Hybrid detection combined that of pattern based and machine learning methods was explored by Nasereddin et al [3] Vulnerabilities in code can be identified by the static analysis, while the dynamic analysis can detect the attacks the runtime. Hybrid approaches achieved 85–95% detection rates. Pattern matching alone cannot be the sole solution for unknown threat. The necessary multi-layered detection frameworks were suggested by the study. Recio et al [19] evaluate Decision Trees, Random Forests and SVMs for SQLi detection within the explainable AI context. The Random Forests had 96% accuracy and gave interpretable classification decisions. The explanation helps the cybersecurity teams trust the AI recommendations. Complex models may reduce transparency. An advisory was made towards a balance between model performance and interpretation. Mustapha et al [20], have studied the detection of SQLi on e-commerce using Random Forests, reaching 97%. SQLi is a risk of data breach and financial loss in e-commerce systems. The reliability improved due to the ensemble nature of Random Forests compared to the single classifiers. The problem is that using only one dataset would limit model generalization. It was suggested that ensemble learning is combined with security policies. CCBA Deep learning model for detecting command injection attack was proposed by Wang et al. [21]. The model combined CNN and BiLSTM to obtain a 99.3% accuracy in the detection. While the method is focused on command injection, it is applicable to SQLi attacks. However, the strength of deep learning in cybersecurity was stressed in this study. However, there remains a challenge to high resource demands. Alghawazi et al [7] investigated the application of the ML models ANN, DT, SVM, or RF in SQLi detection. Once trained, complex SQL query structures could be learned with 95.6% accuracy by ANNs. In addition, to improve performance, they suggested that ANN could be hybridized with ensemble models. The threat keeps evolving, and thus, continuous retraining is required to adapt to that. It was observed that there was criticality in AI driven detection for robust cybersecurity. For monitoring SQLi, machine learning is merged together with automated scanning in the work of Cahyadi et al [34] A 92 percent detection rate was achieved. Adaptness was enhanced by machine learning, and automation moves quickly towards threat identification. The points that were noted included the time it takes to deploy in real time and the cases of high false positives. Instead, they convinced people to use ML in automated security workflows.

To improve SQLi detection, Odeh et al [24] use boosting ensembles of learning methods. The reduction in false positives were achieved at the expense of a 96% detection accuracy. Nevertheless, ensemble techniques incur in the cost of the increase of computationally complexity. The researchers also described how combining multiple models improves resilience. Security systems were recommended to be strengthened by the use of ensemble learning. Shahbaz et al [23] applied CNNs to SQLi with an accuracy of 97.41%. CNNs were able to learn effective structured SQL query patterns for detection of attacks. In fact, they also adapted well to novel attack variations. Nevertheless, CNNs required large datasets as well as extensive training resources. Such a deep learning detection enhancement was supported by their study. Paul et al [27] presented a CNN LSTM based SQLi detection and prioritization, named SQLR34P3R. Using real world traffic datasets the model got 97% F1 score. It allowed resource allocation to be improved by risk based vulnerability prioritization. For smaller organizations, a challenge was high computational costs. According to their study, one way to defend SQLi would be to integrate ML and real-time monitoring. Kapoor et al [32] proposed a token occurrence analysis method for SQLi detection. Their approach achieved 96.6% accuracy by analyzing dynamic SQL query structures. It performed better than traditional blacklisting techniques against novel attacks. However, real-world testing and scalability challenges remain. Further research on optimization and integration was recommended. According to Johny et al [33], It classifies the methods of preventing SQLi into machine learning, cryptography and pattern matching. Strong defence in cryptographic methods was provided, but brought performance overheads. In order for ML based detection to be effective, large, labeled datasets needed to be available. Known threats were matched against pattern, but obfuscation was not. Its defense was suggested to be integrated multimethod. In particular, in Zhang et al [26], they identified vulnerabilities in Natural Language Interfaces to Databases (NLIDBs). Finally, they showed how to use TrojanSQL attacks to change text to SQL parsers with 99% such success rates. These advanced injection methods proved to be greater than existing defences. We recommended secure development practices and practicing very strict schema enforcement. Security of NLIDB was a point for further research.

Table 2. Comparative Analysis of SQL Injection Detection Techniques from Reviewed Studies.

Authors	Limitations	Mitigation Approaches	Advantages	Disadvantages	Accuracy (%)	Key Findings
Hirani et al. [12]	Limited dataset scope	CNN-based detection	High accuracy,	Computationally expensive	98.6%	CNNs outperform

			AI-driven approach			other models in SQLi detection.
Raut et al. [14]	Limited real-world testing	Aho-Corasick algorithm	Detects malicious SQL queries effectively	May miss advanced attacks	91.3%	Pattern matching helps identify SQL injection attempts.
Kakisim et al. [16]	High computational requirements	LSTM-CNN deep learning model	Highest accuracy among models	Resource intensive	99.96	Multi-view deep learning significantly improves SQLi detection.
Augustine et al [18]	Low adoption rate in practice	Machine learning models	AI-driven security improves detection	Implementation challenges	80%	AI improves SQLi detection but remains underutilized
Recio et al [19].	Explainability challenge	Decision Trees, RF, SVM	High accuracy	Complexity in real-time use	96%	Explainability improves trust in ML models
Mustapha et al [20]	Feature selection issues	Random Forest, ANN	Strong classification performance	Single dataset used	97%	Random Forest is highly effective
Wang et al [21]	Focuses on command injection only	CNN, BiLSTM	High detection rate	Limited SQL-specific discussion	99.3%	Deep learning enhances detection accuracy
Nasereddin et al [3]	Limited dataset diversity	Hybrid detection methods	Covers 60 papers	Lack of real-world evaluation	85-95%	Case-based Explanation of Classification Models
Alghawazi et al [7]	Limited evaluation of non-ML techniques	Artificial Neural Networks (ANNs)	High accuracy in intrusion detection	Requires extensive training data	95.6%	ANNs outperform other ML models for detecting SQL injections
Cahyadi et al [34]	Limited real-time deployment analysis	Machine Learning + Automated Scanning	Effective in real-time detection	May have high false positives	92%	Automated scanning combined with ML improves SQLi detection
Kapoor et al [32]	No real world implementation or large scale testing	Token-based query validation, bit occurrence analysis	High accuracy (96.6%) in identifying SQL injection patterns	Does not test against obfuscated or advanced SQL injection techniques	96.6%	Proposes a novel bit occurrence analysis method for SQL injection detection.

Odeh et al [24]	Potential for model complexity	Ensemble Learning	High robustness against false positives	Increased computational cost	96%	Ensemble learning improves detection reliability and reduces false positives
Johny et al [33]	No empirical comparison of methods in real-world scenarios	Cryptography, machine learning, pattern matching (AhoCorasick, Knuth-Morris-Pratt)	Highlights advantages and drawbacks of different techniques	Machine Learning based methods require extensive datasets for training	97.41%	Compares cryptographic, machine learning, and pattern matching techniques for SQL injection prevention.
Shahbaz et al [23]	Computationally expensive	Convolutional Neural Networks (CNNs)	Learns structured attack patterns effectively	Requires large datasets	97.41%	CNNs are highly effective in learning SQLi attack patterns

6. Detection Techniques for SQL Injection Attacks

6.1 Log Analysis

The detection of SQL Injection (SQLi) attacks is a fundamental technique that consists on log analysis, this consists of running through the server logs, as these are provided by web and database systems to identify unexpected or malicious behaviour suggestive of injection attempts. They are rich sources of forensic evidence in these logs: the request details, the timestamps, user input, and executed queries. The advantage of log analysis is the fact that it is retrospective — administrators can trace the path of the attack, identify compromised inputs and correlate suspicious pattern with other system events. For example, the repeated requests such as SQL syntax (i.e., ' OR '1'='1) may be indicating brute force probing or reconnaissance attempts of attackers. In practice, log analysis may be done manually or automatically. Labor-intensive and error prone, manual study is too much work and errors can easily get in. Instead, automated log analyzers are able to look for suspicious patterns in live (as well as historical) systems. The tools use rule based algorithms or machine learning classifiers to detect known signatures or statistical anomalies. This allows also for reflection of specific application contexts or threat models. Potential breaches can be notified to system administrators as real time alerts so that they can get to incident response faster.

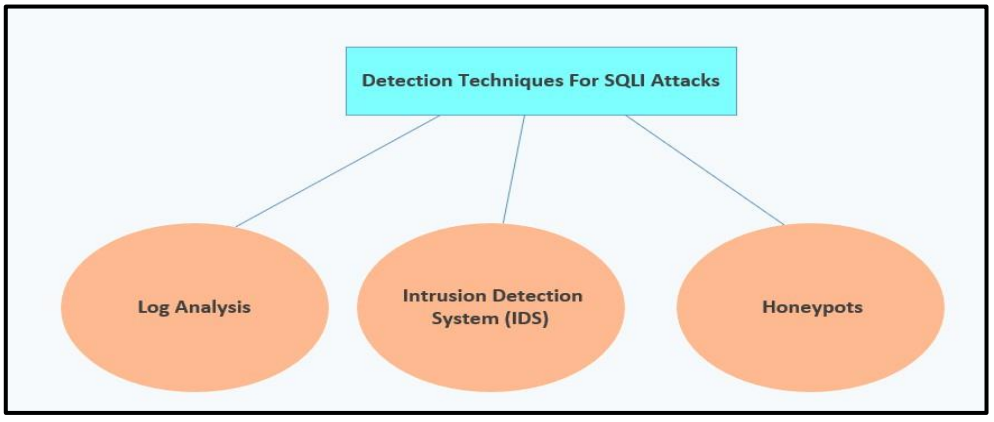


Figure 6. Overview of Detection Techniques Used to Identify SQL Injection Attacks.

However, log analysis is not problem free. The systems may also suffer from large amounts of data which would overwhelm and result in missed detections or false negatives. In addition, attackers may try to delete or alter the logs so to cover their tracks. Because of this, these secure log storage and integrity verification mechanisms (such as checksums or digital signatures) are a requirement. However, log analysis is most powerful when combing with it other detection methods, and then as a part of a defence in depth strategy intended to mitigate sql injection attack Hadabi et al. [35].

6.2 Intrusion Detection Systems (IDS)

In practice, this means that intrusions detection system (IDS) is a key part of defending against real time SQLi attacks. They are network traffic or host activity monitoring systems in order to observe malicious actions. IDS it has three main ways of identification: signature-based, anomaly based and behavior based detection. Instead of comparing incoming traffic against a database of known SQLi patterns (such as strings UNION SELECT or –) as signature based IDS does, this IDS signatures the incoming traffic for a list of patterns and characters. This is an effective and powerful attack against previously encountered attacks, but it is overwhelmed by zero day threats. Therefore, anomaly based IDS has its baseline of normal behavior to alert any deviation and can detect novel attacks. Behavior based IDS take a step further by developing model of the normal application behavior and not identifying deviations in the way how SQL queries are written or executed. For example, at a user account that usually runs read only queries, suddenly begins to modify the database schema, this would be flagged as a suspicious behavior by a behavior based IDS. In ubiquitous computing systems, these systems usually make use of machine learning algorithms that run in a continuous manner in trying to understand what normal vs. malicious are. There are host based (HIDS) and network based (NIDS) IDS solutions that can examine information passing in and out of the server, through application logs or via system calls. Comprehensive threat detection is provided by both forms. Whatsoever, anomaly and behavior based IDSs suffer from high false positive rate. They can blind or make administrators insensitive to the actual threats. Furthermore, sophisticated attackers can craft low and slow SQLi payloads in order to avoid detection of their payload. An IDS implementation should therefore be tuned with contextual rules and fed with threat intelligence feed to update its detection capabilities to mitigate this. Using IDS along with SIEM platforms, it provides real time analytics, has the power of historical correlation also brings in forensic insights, making it a very essential component in SQLi defense strategy Manhas et al [36].

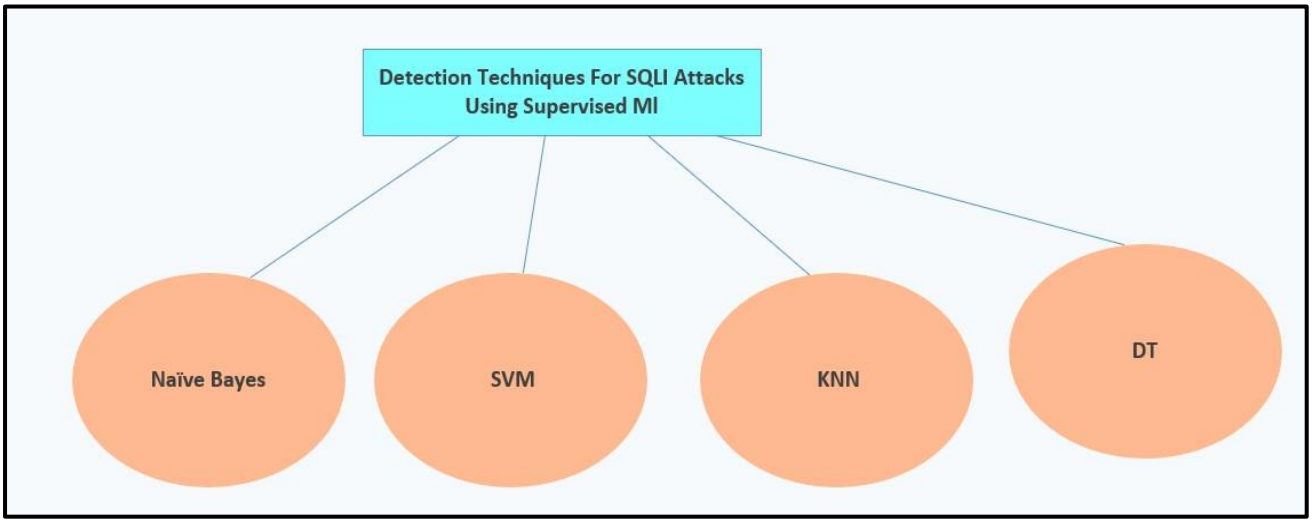
6.3. Honeypots

SQL Injection detection using honeypots is a proactive and deceptive way of detection. They are deliberately vulnerable systems which organizations can use to observe the intrusion tactics without affecting operational systems. In this case, you can record every interaction and every exploit, including attack payload, source IP, and exploitation techniques and record it in a honeypot SQLI web application. This can be used in order to better understand attacker behaviour and the crafting of better defences. Honeypots are not part of the legitimate traffic paths, which means any implication of those honeypots makes the interaction suspicious, thus lowering the false positive rate. In addition, deploying honeypots can also be used as an early warning system. Honeypots allow us to see scanning activities or SQLi attempts even before an attacker gets to production systems and thus provides some valuable lead time for defensive action. High interaction honeypots are closer to the real systems and are more effective on attacks by advanced tactics than low interaction honeypots are, but the latter are easier to deploy, and to capture the usual attack signatures. Honeypot's utility can be extended into production environments without making a real vulnerability vulnerable using honey tokens (fake database entries that inform administrators that someone has accessed them). However, being a honeypot has its benefits, but it can also be a demanding task by nature. Attackers or misusers can be alerted to poorly configured honeypots or it's used for more attack launching. Monitoring and segmentation must be done periodically. Account should also be taken of legal and ethical aspects on the collection of attacker data. Using honeypots strategically they have the power to enhance situational awareness and feed threat intelligence gathered via SQL Injection attempts, both in defence and research Chou et al [37].

6.4 Detection of SQL Injection Attacks Using Supervised Machine Learning and CNN Models

The web application threat SQL Injection (SQLi) persists strongly which requires innovative detection systems to identify and counteract attacks during their execution. The use of traditional signature-based and heuristic defence techniques fails to adapt effectively against changing attack patterns. The authors Tung and Tean Thong [38] introduced a hybrid SQLi detection framework which combines supervised machine learning algorithms with Convolutional Neural Networks (CNNs) for dynamic real-time assault discovery along with classification operationality. The system advances detection accuracy through the combination of Naïve Bayes and Support Vector Machine (SVM) and K-Nearest Neighbors, (KNN)

and Decision Tree (DT) approaches which bring distinct strengths to the system. As shown in the Figure 7, detection of SQL injection attacks using supervised machine learning and CNN Models.



6.4.1 Naïve Bayes

Naïve Bayes serves as a probabilistic classifier which implements Bayes' theorem and makes the assumption that features remain conditionally independent from one another. The SQLi detection process benefits from this approach because it has reduced computational complexity and quick classification timing. The algorithm performs exceptionally well with datasets of various dimensions so researchers can analyze extensive SQL data patterns. Gaussian Naïve Bayes demonstrates success as a classification approach because it accepts the normal distribution of query behavior features that stem from continuous numerical data measuring length, keyword density and entropy Tung and Thong [38]. The Naïve Bayes excels at achieving good performance results with minimal training data during the detection of SQLi attacks. Naïve Bayes provides fast training with minimal resources which makes it fit perfectly for real-time applications as well as limited-resource environments. The model demonstrates strength against irrelevant features which decreases the risk of performance reduction when non-discriminatory characteristics exist in the input information. The system exhibits enhanced generalization features when processing unfamiliar types of SQLi variants due to its inherent robustness. The Naïve Bayes model suffers from reduced accuracy because it assumes independence between features despite the common situation where SQL query features depend on one another within their contextual environment. Jason and Asha added Naïve Bayes as an element in their combined ensemble framework that contains sophisticated modeling approaches. Naïve Bayes benefits the system with its fast performance yet more complex models backs it up by handling intricate feature dependencies which the Naïve Bayes model lacks ability to capture. The hybrid system manages to achieve high efficiency alongside acceptable detection accuracy in practical use scenarios Tung and Tean Thong [38].

6.4.2 Support Vector Machine (SVM)

The supervised machine learning algorithm Support Vector Machine (SVM) performs exceptionally well at two-class classification because it works best with data which displays distinct separation boundaries. The detection of SQLi attacks becomes effective through SVM because the algorithm transforms data features into higher dimensions where easy linear separation emerges. SVM allows Jason and Asha's model to create optimal decision boundaries between SQL queries that are malicious and legitimate ones. The algorithm achieves better generalization on unknown attack inputs and decreases overfitting risks by creating the maximum support vector distance Tung and Tean Thong. [38] Projecting features through SVM in high-dimensional spaces shows to be one of its main advantages when analyzing SQLi datasets that contain several syntactic and semantic indicators. Through the kernel trick SVM can discover hidden non-linear patterns in the data without performing explicit transformation measures. Such capabilities are vital to differentiate attacks that use obfuscated SQLi payloads which imitate standard input structures. SVM demonstrates effectiveness in operating with reduced sample sizes thus serving as an ideal method to detect SQLi attacks in real time because quick responses remain essential. SVM systems require strong parameter optimization because they become less accurate when processing highly noisy datasets or classes

that overlap with one another. Training processes require high computational power when dealing with extensive dataset records. Programming within the proposed system merges SVM's exacting precision with other classifiers because SVM possesses robustness. The system uses boundary detection abilities of SVM alongside KNN and CNN models to produce balanced performance when detecting various types of attacks. SVM integration produces a high positive detection capability for detecting hard-to-spot SQLi attacks that escape basic security models Tung and Tean [38].

6.4.3 K-Nearest Neighbors (KNN)

K-Nearest Neighbours (KNN) is a non-parametric instance based learning algorithm for classification of inputs based on labels of their nearest neighbors in the feature space. KNN is beneficial for deteting SQLi because it makes no assumptions regarding the un-derlying data distribution and thus can detect both classic and novel attack patterns only by spatial relationship in the data. KNN is used in the author's framework to uncover the hidden relational structures in the SQL query features, and the proximity based similarities of the malicious and benign input vectors Tung and Tean [38]. KNN is simple and flexible. In particular, it is very useful in exploratory data situations where relationships between features are unclear and are non-linearly model able. The computational burden is shifted to the prediction phase where the training part of KNN is essentially instantaneous and is referred to as the lazy learning approach. However, this design is a double edged sword for real time detection here as setup is fast but inference will need to be optimized (e.g. KD trees, approx. neighbors) to scale well at inference time. This is nevertheless a model that enjoys its having the ability to keep pace with developing SQLi attack strategies without retraining. KNN is however sensitive to imbalanced datasets and irrelevant features that may disturb distance calculations and bias classification accuracy. In order to counter this authors used feature selection and normalization in the feature space to improve the performance of KNN in the ensemble system. Also, as KNN is interpretable, it can be an essential part in the development of diagnosis tools in the form of visualization of decision boundaries, and neighborhood distributions. Second, in the hybrid model, with local decision context, KNN supplements other classifiers, it's a meaningful tool for nuanced SQLi attacks detection Tung and Tean [38].

6.4.4 Decision Tree (DT)

A Decision Tree classifier constructs a tree like model of decisions on the basis of feature values. Each of the internal nodes is a condition, each of the branches an outcome and each leaf node a class label. Decision Trees are good at determining what is SQLi, as they can differentiate between categorical and numerical features, making each value treated as if there is numerically encoded information (which can also be used). The authors' system uses Decision Trees to examine sequential query attributes—a logical operator presence, input length, and suspicious keywords to sequentially screen the classification process using interpretable rules Misquitta et al [39]. The transparency of and interpretability of the Decision Trees are one of the major advantages of it. Admittedly, a specific classification of an IO is easily descended from which this logic resides, and so is very easy to understand and to validate detection results. DTs are therefore ideal for situations where accountability and audibilty are a necessity. In addition, the model can handle missing data and has the non-linear decision making capability in heterogeneous SQLi datasets. DTs are also quickly trainable and deployable, are thus suitable for inclusion in real time monitoring systems. However, as Decision Trees can easily overfit the data, the training data should have very little noise and the classes should not dominate during learning. Due to this, the trees can be biased and tend not to generalize well. To approach this, the authors then applied pruning techniques and ensemble balancing to provide a fair and robust classifier. The Decision Tree when combined with a CNN architecture and other machine learning models provides interpretable insights and also increases the system's responsiveness to various SQLi attack signatures Misquitta et al [39].

7. Prevention Techniques for SQL Injection Attacks

SQL Injection (SQLi) attacks stands as one of the biggest and worst security threats to the web applications. Such attacks can be prevented through a defense in depth that consists of many prevention techniques. The best prevention ways are Input Validation, Parameterized Queries and Stored Procedures, which are discussed in this section. Each of these plays a specific and important role setting malicious user input apart from endangering the underlying database integrity. As shown in the figure 8, the prevention techniques for SQL injection attacks.

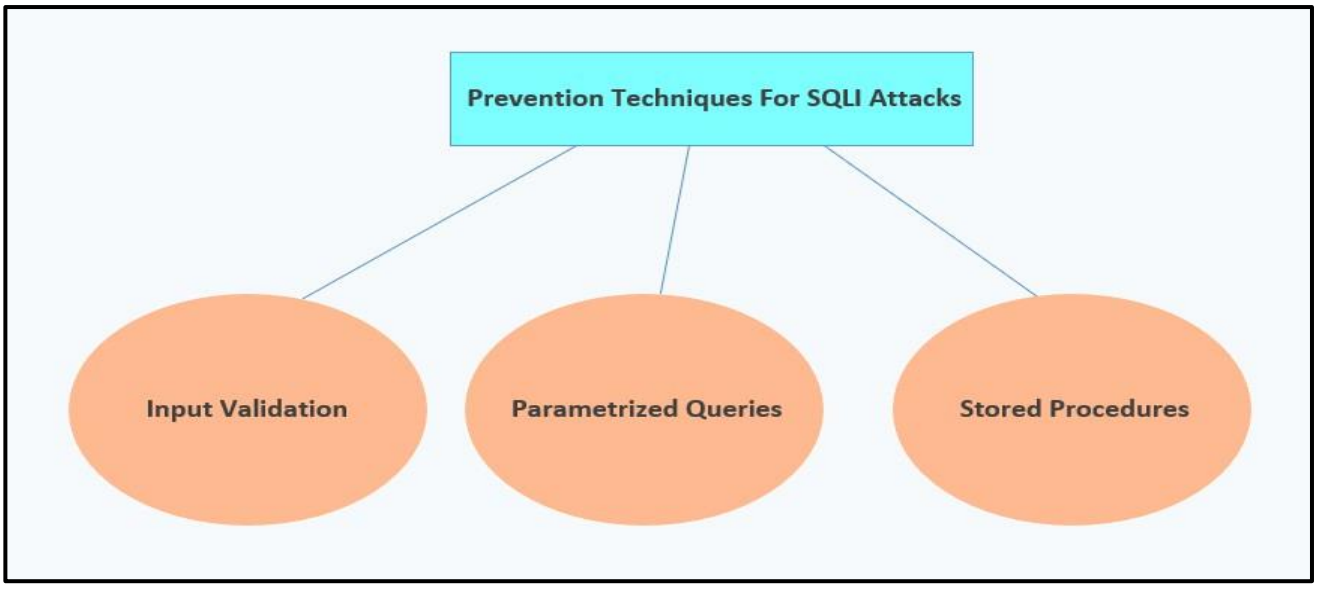


Figure 8. Overview of Prevention Techniques against SQL Injection Attacks.

7.1 Input Validation

The first and most important line of defense against the SQL Injection attacks is input validation. This is a process that occurs prior to processing of these sets of data, or their embedding into SQL queries. Validation also makes sure input stays within acceptable formats, data types and length constraints, as well as character sets to greatly decrease the possibility of malicious payloads to reach the database engine. For instance, numeric fields must usually be in the strictest check, and should not allow any embedded SQL character such as quotes or semicolons. Blacklisting (blocking known bad inputs) is usually less preferred than whitelisting (allowing only acceptable input) because whitelisting minimizes risk of missing new attack vector Kim and Min Su [40]. The client side as well as the server side should do effective input validation. Even though client side validation offers better user experience and sometimes will prevent some malformed input at an early stage, it is inherently untrustworthy because attackers can bypass it with tools such as intercepting proxies or even directly performing an HTTP request to the backend. Therefore, Security Enforcement must take place on server side. Additionally, the validation process should filter out single quotes (’), double quotes ("), semicolons (;), SQL keywords such as DROP, SELECT, and INSERT, among others, if such inputs are not required in the application’s context. A sanitization library and regular expressions can also help developers ensure that the validation is the same throughout your input fields Kumar and Mohit. [41].

Nevertheless, it alone is not foolproof. Filters are still being successfully bypassed from one attack vector to another, and attackers continue to change their ways to overcome the filters, especially those based on outdated blacklist or filters implemented in a very poor way. Encoding payloads in Unicode or hexadecimal will bypass filters that simply decrypt payloads and only check non escaped input. This emphasizes the need for additional protective layers in forms like shaping the input to validate and also the execution itself should be secure and query separation. Input validation while not an absolute security the application can get a good degree of protection against SQLi by imposing strict constraints on data and acts as a data filtering barrier from malicious input Baklizi et al. [42].

7.2 Parameterized Queries

One of the most popular Prevention strategy against the SQL Injection is the Parameterized queries, also known as prepared statements. Essentially, they separate the SQL code from the user input and so keep user provided data as values, and not executable code. In contrast to dynamic SQL queries, where their input is strung together into the query string, parameterized query uses placeholders (e.g.,?, @param) to insert through user data. This structural separation prevents SQL control injection as whichever the input is, the database engine treats it as literal only Shandilya et al [43]. Parameterized queries are implemented and supported across almost all major programming languages and frameworks including PHP (PDO), Java (JDB), Python (sqlite3 or postpsql2c), .NET (.NET (SqlCommand w parametter)). This widespread support makes it easy to go on to integrate with modern development environments without having to

learn specialized knowledge. This also enables query performance to be enhanced by execution plan caching as query structure is the same even when the parameter values change. This also enhances the code readability from parameters and logic, since parameters and logic clearly defined and isolated Baklizi et al [42]. Parameterized queries have their advantages however, they need to be configured properly to realize this full effect. So, developers should never mix them with dynamic SQL or corrupted parameters, especially in complex operations such as LIKE clauses or IN statements. Additionally parameterized queries prevent SQLi in query structure, but don't prevent data from being sanitized for other vulnerabilities e.g. XSS or logic flaw. This therefore implies that their use should be accompanied by excellent input validation and output creation to be able to provide full spectrum security. However, if parameterized correctly, parameterized queries reduce the attack surface drastically and are recommended by standard of secure coding Yadav et al [44].

7.3. Stored Procedures

Precompiled SQL routines in the form of stored procedures are stored and executed directly within the database server. It helps as a critical mitigation strategy for SQL Injection attack by wrapping SQL logic inside some controlled environment that reduces the necessity of constructing SQL command dynamically at application code. Stored procedures inherently separate data from code by requiring inputs to be passed as parameters, which prevents malicious input from altering the structure of queries in a rude manner. It isolates the data processing logic from the application interface and limits the direct manipulation capability by the external inputs. Moreover, stored procedures provide performance benefits in addition to increased security. Since they are created and compiled and optimized by the database server on creation, repeated execution incurs no parse or plan overhead. So, they are meant for use performing frequently database operations like getting user profile or processing transactions. In addition, stored procedures can also prevent access control and business logic constraints at the database layer instead of the application layer, if the application logic is compromised. For instance, a procedure may prohibit a user from performing DELETE operations, although such capability is unintentionally available at the application level Yadav et al [44]. Similar to all security mechanisms, stored procedures are not immune to abuse. In addition, procedures that concatenate user input or that do not validate parameters can still be susceptible to injection in badly written procedures. In addition, inverting too much on stored procedures will not be code portable and make the systems more complex. To prevent these risks, developers should practice secure coding, parameterization of procedures very strictly, and regularly audit their logic for vulnerabilities. Along with input validation and parameters queries, stored procedures present another layer of defense and assist in forming a sound robust multi-tiered SQL Injection prevention strategy.

8. Future Direction and Recommendations

SQL injection attacks (SQLIAs) pose a serious threat which keep evolving in complexity; thus, they require advanced, adaptive and multi-layered defense approach. While this study details a comprehensive survey of standard and AI driven treatment techniques, there are numerous spaces for productive future work. Aside from that, one major direction lies in integrating XAI function into SQLIA detection frameworks. On the other hand, machine learning and deep learning models such as CNNs and RNNs achieve high detection accuracy on the problem space, but they do not provide interpretability which would allow for trust and usability in real world security operations. Such AI models should first focus on providing interpretable AI that not only detect anomalies but also explain the rationale of making a classification decision. It would obviate the anxiety on the part of administrator and enable better decision making in incident response Recio et al [19].

The second key area for advancement is the detection and defense against SQLI threats on new data platforms, particularly NoSQL databases and other Neural Language Interfaces to Databases (NLIDBs). This paper highlights several such vulnerabilities imposed by these systems which may be ignored by the traditional security models. Security framework needs to be comprehensive in nature that counteracts the semantic parsing problems associated with NLIDBs, and the non-relational query structures of NoSQL. Future systems must deal with queries within the context aware models which understand the query intent and are able to detect in that natural language input injected queries Zhang et al [26].

While this study contributes to the ongoing studies about SQLIA by means of a multi-dimensional review of current best practices followed by a roadmap for providing practical road mapping of traditional defenses, pattern based detection, cryptographic safeguards, and advanced ML based approaches, respectively, as a unified defense strategy. In addition, such a cross domain federated learning system is also required that can share knowledge across the organizational boundaries while ensuring data privacy. Such systems may strengthen collaboration during a defense and maintain regulatory compliance. There is finally a need for hybrid and self-sufficing frameworks with static code analysis, real time behavioral monitoring, and a threat intelligence feed. The detection should be based on adaptive learning these systems should be able to learn in production environments, or automatically updating detection rules given by some feedback loop from the

ongoing threat landscape. Also, we need to rethink the role of honeypots, IDS, and the Web application firewalls (WAF) and include them in the form of a layered and proactive security posture driven by AI tools.

9. Conclusion

SQL Injection Attacks (SQLIA) remain one of the most malicious and prevalent attacks for web applications. Although there have been many innovations in discovering and counteracting such attacks, ranging from secure coding, input validation and parameterized queries, to name a few, attackers across an array of domains keep coming up with new tricks to sidestep the traditional solutions. The mechanisms of SQLIAs were critically reviewed, major attack vectors are classified, and traditional as well as modern countermeasures such as heuristic filters, cryptographic validation, and the latest machine learning and deep learning models were evaluated. Adaptive, scalable, intelligent and adaptive to the changes in the threat landscape, is necessary as the hybrid AI based systems and behavior aware analytics are getting more popular. From the analysis, it has been inferred that Deep Learning models like CNNs, LSTMs, and Hybrid models have shown improved results to detect complex and obfuscated SQLIAs, however, it also poses challenges such as computational complexity, data requirements and explainability. One of the main reasons of future research then is to create lightweight with explanation and real time SQLIA detection models using technologies like federated learning, transfer learning, and adversarial training to make them capable of evolvingthreats without compromising any performance or security. Moreover, they should also focus on securing platforms such as NoSQL databases and Natural Language Interfaces to Databases (NLIDBs) that introduce novel, yet underexplored, vulnerabilities. Finally, SQLIA prevention should be integrated into software development lifecycle. To minimize risks in the writing of code level, organizations must institutionalize regular vulnerability assessments, developer education program and coding in the secure practice of the developer. The best practice of a multi layered defense in depth with static analysis, anomaly detection, IPS and proactive threat intelligence continues to be the most effective. This study highlights not only the gaps in SQLIA defense but also a clear path of future innovating SQLIA defense by keeping performance, scalability, and explainability in mind in the world where everything is more connected.

Corresponding author

MM Hafizur Rahman
mhrahman@kfu.edu.sa

Acknowledgements
The work was supported by King Faisal University, Saudi Arabia.

Funding
No funding.

Contributions
A.L.C; J.S; Conceptualization, S.S; A.A; Investigation, A.L.C; J.S; Writing (Original Draft), S.S; A.A; Writing (Review and Editing) Supervision, A.A; Project Administration.

Ethics declarations
This article does not contain any studies with human participants or animals performed by any of the authors.

Consent for publication
Not applicable.

Competing interests
All authors declare no competing interests.

References

[1] Aliero, M. S., Qureshi, K. N., Pasha, M. F., Ghani, I., & Yauri, R. A. (2020). Systematic review analysis on SQLIA detection and prevention approaches. *Wireless Personal Communications, 112*(4), 2297–2333.
[2] Nair, S. S. (2024). Securing against advanced cyber threats: A comprehensive guide to phishing, XSS, and SQL injection defense. *Journal of Computer Science and Technology Studies, 6*(1), 76–93.

[3] Nasereddin, M., ALKhamaiseh, A., Qasaimeh, M., & Al-Qassas, R. (2023). A systematic review of detection and prevention techniques of SQL injection attacks. *Information Security Journal: A Global Perspective, 32*(4), 252–265.
[4] Chowdhury, S., Nandi, A., Ahmad, M., Jain, A., & Pawar, M. (2021). A comprehensive survey for detection and prevention of SQL injection. In *Proceedings of the 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)* (Vol. 1, pp. 434–437). IEEE.
[5] Lawal, M., Sultan, A. B. M., & Shakiru, A. O. (2016). Systematic literature review on SQL injection attack. *International Journal of Soft Computing, 11*(1), 26–35.
[6] Hlaing, Z. C. S. S., & Khaing, M. (2020). A detection and prevention technique on SQL injection attacks. In *Proceedings of the 2020 IEEE Conference on Computer Applications (ICCA)* (pp. 1–6). IEEE.
[7] Alghawazi, M., Alghazzawi, D., & Alarifi, S. (2022). Detection of SQL injection attack using machine learning techniques: A systematic literature review. *Journal of Cybersecurity and Privacy, 2*(3), 764–777.
[8] Elshazly, K., Fouad, Y., Saleh, M., & Sewisy, A. (2014). A survey of SQL injection attack detection and prevention. *Journal of Computer and Communications, 2014*(3), 1–9.
[9] Ali, A. B. M., Abdullah, M. S., & Alostad, J. (2011). SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks. *Procedia Computer Science, 3*, 453–458.
[10] Gasiba, T. E., Lechner, U., Pinto-Albuquerque, M., & Mendez, D. (2021). Is secure coding education in the industry needed? An investigation through a large-scale survey. In *Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)* (pp. 241–252). IEEE.
[11] Omotunde, H., & Ahmed, M. (2023). A comprehensive review of security measures in database systems: Assessing authentication, access control, and beyond. *Mesopotamian Journal of CyberSecurity, 2023*, 115–133.
[12] Hirani, M., Falor, A., Vedant, H., Mehta, P., & Krishnan, D. (2020). A deep learning approach for detection of SQL injection attacks using convolutional neural networks. *Department of Computer Engineering, MPSTME, NMIMS University*, Mumbai, India.
[13] Hajar, S., Jaafar, A. G., & Rahim, F. A. (2024). A review of penetration testing process for SQL injection attack. *Open International Journal of Informatics, 12*(1), 72–87.
[14] Raut, S., Nikhare, A., Punde, Y., Manerao, S., & Choudhary, S. (2019). A review on methods for prevention of SQL injection attack. *International Journal of Scientific Research in Science and Technology, 6*(4), 1–7.
[15] Shachi, M., Shourav, N. S., Ahmed, A. S., Brishty, A. A., & Sakib, N. (2021). A survey on detection and prevention of SQL and NoSQL injection attack on server-side applications. *International Journal of Computer Applications, 183*(18), 1–7.
[16] Kakisim, A. G. (2024). A deep learning approach based on multi-view consensus for SQL injection detection. *International Journal of Information Security, 23*(6), 1541–1556.
[17] Madhvan, R., & Zolkipli, M. F. (2023). An overview of malware injection attacks: Techniques, impacts, and countermeasures. *Borneo International Journal, 6*(1), 22–30.
[18] Augustine, N., Sultan, A. B. M., Osman, M. H., & Sharif, K. Y. (2024). Application of artificial intelligence in detecting SQL injection attacks. *JOIV: International Journal on Informatics Visualization, 8*(2), 2131–2138.
[19] Recio-García, J. A., Orozco-del Castillo, M. G., & Soladrero, J. A. (2023). Case-based explanation of classification models for the detection of SQL injection attacks. In *Proceedings of the ICCBR Workshops* (pp. 200–215).
[20] Mustapha, A. A., Udeh, A. S., Ashi, T. A., Sobowale, O. S., Akinwande, M. J., & Oteniara, A. O. (2024). Comprehensive review of machine learning models for SQL injection detection in e-commerce. *World Journal of Advanced Research and Reviews, 23*(2), 451–465.
[21] Wang, X., Zhai, J., & Yang, H. (2024). Detecting command injection attacks in web applications based on novel deep learning methods. *Scientific Reports, 14*(1), 25487.
[22] Alwan, Z. S., & Younis, M. F. (2017). Detection and prevention of SQL injection attack: A survey. *International Journal of Computer Science and Mobile Computing, 6*(1), 5–17.
[23] Shahbaz, M., Mumtaz, G., Zubair, S., & Rehman, M. (2024). Evaluating CNN effectiveness in SQL injection attack detection. *Journal of Computing & Biomedical Informatics, 7*(1), 1–10.
[24] Odeh, A., & Taleb, A. A. (2024). Ensemble learning techniques against structured query language injection attacks. *Indonesian Journal of Electrical Engineering and Computer Science, 35*(2), 1004–1012.
[25] Yunus, M. A. M., Brohan, M. Z., Nawi, N. M., Surin, E. S. M., Najib, N. A. M., & Liang, C. W. (2018). Review of SQL injection: Problems and prevention. *JOIV: International Journal on Informatics Visualization, 2*(4), 215–219.
[26] Zhang, J., Zhou, Y., Hui, B., Liu, Y., Li, Z., & Hu, S. (2023). Trojansql: SQL injection against natural language interface to database. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (pp. 4344–4359).
[27] Paul, A., Sharma, V., & Olukoya, O. (2024). SQL injection attack: Detection, prioritization & prevention. *Journal of Information Security and Applications, 85*, 103871.
[28] Abdullayev, V., & Chauhan, A. S. (2023). SQL injection attack: Quick view. *Mesopotamian Journal of CyberSecurity, 2023*(1), 30–34.
[29] Alenezi, M., Nadeem, M., & Asif, R. (2021). SQL injection attacks countermeasures assessments. *Indonesian Journal of Electrical Engineering and Computer Science, 21*(2), 1121–1131.
[30] Kareem, F. Q., Ameen, S. Y., Salih, A. A., Ahmed, D. M., Kak, S. F., Yasin, H. M., Ibrahim, I. M., Ahmed, A. M., Rashid, Z. N., & Omar, N. (2021). SQL injection attacks prevention system technology. *Asian Journal of Research in Computer Science, 6*(1), 13–32.
[31] Mane, S. B., Kakade, K. S., Shingare, S., & Halgare, N. M. (2024). SQL injection authentication security threat. *International Journal of Electronic Security and Digital Forensics, 16*(5), 474–485.
[32] Kapoor, A. (2023). SQL-injection threat analysis and evaluation. *SSRN*. https://doi.org/10.2139/ssrn.4430812

[33] Johny, J. H. B., Nordin, W. A. F. B., Lahapi, N. M. B., & Leau, Y. B. (2021). SQL injection prevention in web applications: A review. In *Advances in Cyber Security: Third International Conference, ACeS 2021, Penang, Malaysia, August 24–25, 2021, Revised Selected Papers* (pp. 568–585). Springer.
[34] Cahyadi, N., Yutia, S. N., & Dorand, P. (2023). Enhancing SQL injection attack prevention: A framework for detection, secure development, and intelligent techniques. *Journal of Informatics and Communication Technology (JICT), 5*(2), 138–148.
[35] Hadabi, A., Elsamani, E., Abdallah, A., & Elhabob, R. (2022). An efficient model to detect and prevent SQL injection attack. *Journal of Karary University for Engineering and Science, 2022*(1), 1–10.
[36] Manhas, S. (2022). An interpretive saga of SQL injection attacks. In *Emerging Technologies in Data Mining and Information Security: Proceedings of IEMIS 2022, Volume 1* (pp. 3–12). Springer.
[37] Chou, D., & Jiang, M. (2021). A survey on data-driven network intrusion detection. *ACM Computing Surveys, 54*(3), 1–36.
[38] Tung, T. T. (2024). Detection of SQL injection attack using machine learning (Doctoral dissertation, UTAR).
[39] Misquitta, J., & Asha, S. (2023). SQL injection detection using machine learning and convolutional neural networks. In *Proceedings of the 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT)* (pp. 1262–1266). IEEE.
[40] Kim, M. S. (2022). A study on the attack index packet filtering algorithm based on web vulnerability. In *Proceedings of the IEEE/ACIS International Conference on Big Data, Cloud Computing, and Data Science Engineering* (pp. 145–152). Springer.
[41] Kumar, M. (2022). SQL injection attack on database systems. In *Wireless Communication Security* (pp. 183–198). Springer.
[42] Baklizi, M., Atoum, I., Hasan, M., Abdullah, N., Al-Wesabi, O., & Otoom, A. (2023). Prevention of website SQL injection using a new query comparison and encryption algorithm. *International Journal of Intelligent Systems and Applications in Engineering, 11*(3), 228–238.
[43] Shandilya, S. K., Ganguli, C., Izonin, I., & Nagar, A. K. (2023). Cyber attack evaluation dataset for deep packet inspection and analysis. *Data in Brief, 46*, 108771.
[44] Yadav, N., & Shekokar, N. M. (2022). SQL injection attacks on Indian websites: A case study. In *Cyber Security Threats and Challenges Facing Human Life* (pp. 153–170). Chapman & Hall/CRC.