# SQL Injection Attacks(SQLIA) detection and prevention methods using machine learning and traditional approach (A Comparison study)

**Fouad Raheem Abdulhamza[1], Dr. Rana JumaaSurayh Al-Janabi[2]**
**[1]University of Al-Qadisiyah, Department of Computer Science, Qadisiyah, Iraq**
fouad.raheem@qu.edu.iq, rana.aljanaby@qu.edu.iq

## ABSTRACT

This sort of intrusive attack on web-based applications is particularly serious since it might expose the secrets and safety of data. Illegal individuals get access to the web-based database and the data it contains by stealing it. There are a variety of ways to detect and prevent this form of attack, but they are not sufficient since many of them do not work for all types of assaults. In this research, many types of SQL Injection attacks were examined as existing methods of detecting and preventing them. Because of the current techniques of preventing From the user end. The research Comparison used machine learning and the traditional approach to detecting and preventing them and developers would have to create separate validation methods for every web page that obtained data from the server.

Keywords: SQLI, WA, machine learning, traditional approach, detection, prevention



**DOI Number:10.14704/nq.2022.20.8.NQ44797** **NeuroQuantology 2022; 20(8): 7715-7725**

## 1- Introduction

In a relational database, SQL (Structured Query Language) is a relational database language that is used to insert, update, remove, change, as well as query data in an organized fashion. As far as a database is used by the web application, the vast majority of its interactions with the database are accomplished using SQL statements. A major contributing factor to the existence of Threats related to SQL injection is that when programmers write code, They make use of text concatenation in order to construct SQL statements, which are subsequently sent to the database for processing.Consequently, attackers may alter the SQL comments by SQL phrases or special symbols in the SQL query. Attack on the system occurs as a consequence of the procedure's execution. The main cause is to insert confidence in user-submitted data, without even a data filter from the user and without an acceptable level of verification on the server's part, in order to accomplish the attacker's desired objective, which may include stealing important system data and gaining server control. The fundamental ideas and methods of an SQLIA are seen in Figure 1.[1]

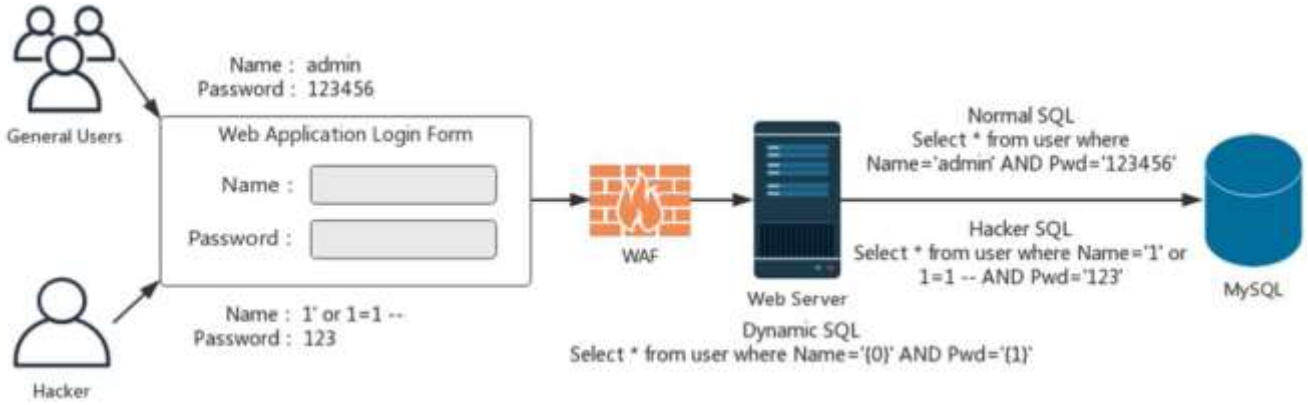


Fig1. SQL Injection Attack Mechanism

SQL Injection attacks on websites, which were present for a long period of time, are an increasingly harmful source of personal data theft, as well as a source of severe economic consequences for enterprises and organizations[2]. This is especially true when existing threats are changed and developed, and new attack mechanisms continue to arise and become more sophisticated. It is estimated that industry and security companies dedicate a significant amount of services devoted to the reduction of internet attacks, as well as a variety of current reduction strategies have limits that scientists are continuously working to overcome [3].

Signature detection, often known as static analysis of incoming web traffic, is a technique for identifying patterns in data and is used to mitigate a large proportion of classic web attacks. An important part of this technique is the development of an attack signature that can be detected by a firewall or other performance rating. The suspect traffic may subsequently be blocked by the firewall or by other threat security cooperation mechanisms if a signature is identified. This technology has the advantage of being rapid and being able should be carried out in real-time in order to protect network resources; however, one disadvantage is that it can only detect known threats, which is a drawback .

It is also possible to use a strategy for web attack mitigation that is specifically targeted at SQL injection. A faulty SQL query will be flagged by this technique if it is recognized in the structure of the incoming SQL queries. The attack is classified as an SQLIA. This approach has strong detection outcomes and can also identify novel attacks that utilize faulty queries; however, It has a flaw in that it requires a substantial understanding of the use and structure of "normal" queries, which is not always the case .

SQL injection detection with machine learning(ML), which is currently under investigation, uses ML techniques to identify SQL injection attacks. Support vector machines (SVM), Decision trees (DT), rule-based learning approaches, and convolutional neural networks (CNN) are all popular strategies in this field of study .

One of the most significant advantages of these techniques is their ability to identify new attacks as they occur. A significant disadvantage of these strategies, Although it is possible to boost processing time, this varies depending on the algorithm and datasets that are being used[4].

**2-SQL INJECTION**

A hacker inserts SQL code into a web form's login box in order to make changes to the database. Assaulters may discreetly conduct malicious actions on a web application's core database using SQL injection attacks .[5]

**2.1Types of SQL injection**

**2.1.1Tautologies.**

This is a popular injection attack in which hackers rewrite the query in such a manner that it always returns true after it has been executed. They have the



*e*ISSN1303-5150

ability to log in as administrators or even completely fake individuals. Hackers may get access to any information by injecting code into a conditional expression .

SELECT * from table WHERE table id = 123;

After injecting or 1=1

SELECT * from table WHERE table id = 123 or 1=1

Here, The hacker will be able to readily access all of the information about the table associated with this exact table id.

**2.1.2Piggyback Queries.**

In this injection attack, one query is implanted into another query. In the tailing queries, there are harmful queries. A subquery of the main query is run as part of the main query.

SELECT * FROM item WHERE item id = 123 or 1=1;

DROP TABLE item;

Here, the ';' character marks the end of one query and the start of another.

**2.1.3Alternate Encodings**

In this approach, hackers combine the function "CHAR" with numerical coding in order to retrieve the character that was originally entered. Since char(47) is not the same as the letter '/',they may avoid filtering out this unwanted character by using char(47)[6].

**2.1.4 Illegal or Logically incorrect query.**

Cyber attackers execute logically wrong queries that cause the system to make errors that provide debugging information. This provides the attacker with the ability to obtain injectable parameters. The query in the given description generates a database error message, which contains information about the database itself.[7]

SELECT accounts FROM users WHERE login=' ' AND pass=' ' AND

pin= convert ( int , ( select top 1 name from sysobjects WHERE xtype='u'))

Error Message

"Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value 'Cred-itCards' to a column of data type int"

**2.1.5 Union query**

Hackers utilize the UNION operators in conjunction with a standard query to insert malicious inquiries into the system. The tailing query in the example below is malicious because it has the capability of obtaining private information while avoiding the authentication procedure.[8]

SELECT * from table_ producer WHERE id='167' UNION select * from credit_card1 WHERE user = 'admin'--' and pass='password'

**2.1.6 Stored Procedure.**

The hacker tries to use malicious inputs to run stored Procedures that are already existent in the database(DBMS). A database management system (DBMS) is often where procedures that enhance the operation of the database are kept and allowed to interact with them. Essentially, stored procedures are a collection of codes that allow you to accomplish certain actions without having to build them from scratch every time. They include certain potentially dangerous codes that attackers may use to get access to the system.

CREATE PROCEDURE database (DB) Name .is Authenticated

@user Name varchar2, @pass varchar2, @pin

int AS EXECv ("SELECT acont FROM t_ producer WHERE login ='" + @user Name + If' and

pass=' " +@password+ and pass=" +@pass);



The use of a stored procedure that has been approved returns true or false depending on whether it was authorized or unauthorized. If the attackers provide the input as ' '; SHUTDOWN; for the user_name and pass, the Stored Procedure returns the following query expression, which causes the system to be shutdown [9].

SELECT User_name FROM Table_user WHERE User_name = user_1 AND pass=' '; SHUTDOWN;

**2.1.7 Inference.**

Using Inference attacks, an attacker may change the database or web application's behavior. There are two approaches to this type of attack, which are blind and timing attacks, respectively[10].

**2.1.7.1 Blind Injection :**

The kind of SQLIA arises when the source code fails to hide an error message, resulting in the relational database application being vulnerable. The error messages aid SQLIA in compromising the relational database by requesting a sequence of logical inquiries using SQL commands, which the SQLIA can then answer. The blind injection assault is shown in the following [11].

SELECT password FROM user_Table WHERE username= 'user a'and 1 =0 - AND pass : AND pin= O

SELECT information FROM user_Table WHERE username= 'user a' and = 1 - AND pass : AND pass= O

**2.1.7.2 Timing Attacks:**

When an attacker observes database reply timing delays, the attacker steals information from the database. This kind of attack made use of an if condition statement in order to accomplish the goal of delaying time. In the branches, there is a term called WAITFOR that forces the relational database to postpone its answer by a certain time.

The interesting timing attack query.[11]

declare @ varchar (6000) select @s = database _name 0 if (ascii (substring (@s, 1, 1)) & (Power (4, 0))) > 0 waif for delay '0:0:9'

**3. RISK ASSOCIATED WITH SQL INJECTION ATTACKS (SQLIA) :**

Databases are vulnerable to SQL injection attacks, and the hazards that come with them drive hackers to target these systems. The main effects of these flaws are exploited by attacks on the following elements:[12]

**3.1 Authorization**

When a successful SQL Injection Attack is carried out on a susceptible SQL database, the critical and important data that is stored in the database may be altered.

**3.2 Authentication**

Due to a lack of sufficient control over the input fields on the authentication page, it the feasible to log in as a regular user to a web application without knowing the authorized user's password.

**3.3 Confidential**

databases include sensitive data such as personal information, credit card numbers, and social security numbers. Consequently, the loss of a breach of confidence is a significant issue associated with SQL injection attacks (sqlia).

**3.4 Integrity**

A successful SQLIA allows an attacker not just to get access but also to change or even destroy sensitive information.

**3.5 Database Fingerprinting**

A database-specific attack may be employed if the attacker knows what kind of database is being used in the backend. The purpose of database-specific attacks is to take advantage of weaknesses in a certain DataBase Management System (DBMS).

**4 .PROBLEM BACKGROUND SQLIAs**

developers who were victimized by the SQL injection did not consider security during the initial design phases of the system. Developers have a propensity to think that the functioning of a system will always proceed in the manner that has been developed and planned in advance of time. This often enabled them to incorrectly deal with challenges, which in the majority of cases resulted in the attacker launching a successful SQLIA against the target system. In a genuine SQL statement, the attacker transfers harmful SQL codes to website apps, datasets, and other resources, and the malicious SQL code provides various error messages based on the problem statement that was utilized. SQL injection attacks would take advantage of any discrepancies between the answer and the results or error messages .

Although there are numerous SQLIA detection and Prevention Strategies that have been developed by a variety of experts in order to prevent SQLIA, a large number of database(DB) security experts spend the majority of their time deciding which Strategies are the most effective in order to prevent SQLIA, which results in a significant amount of money, time work, and power being wasted. In order to avoid SQLIA based on patterns that are recognized, many database security specialists use Intrusion Prevention Systems (IPS) or other kinds of firewalls. However, this solution is very expensive, which most small and medium-sized businesses cannot afford .



In most cases, SQL Injection Attacks(SQLIA) is the consequence of insufficient attention to database security on the part of third-party programmers, or the programmers themselves lack understanding of effective methods and approaches for preventing SQLIAs. The paper "A Survey on Detection and Prevention of SQL Injection Using Machine Learning" was written in order to make things easier for programmers and database security experts when it comes to SQLIA detection and Prevention Techniques . [13]

**4.1Issues and challenges**

Unauthorized access, modification, and deletion of database material as a result of an adversary's SQL injection assault

**4.1.1Defects in the Design**

User input on a website is not validated due to the absence of a checking feature.

SQL Injection Prevention Technique is not implemented at the CGI layer .

Allows results of user input to be transmitted to the database layer, changing the SQL statement format .

**4.1.2Coding Weaknesses**

The trust user input and not sanitize information received during the SQL statement rebuilding procedure
Building SQL statements that will be submitted to the database via the use of concatenation of inputs
Accepting input from stored procedure instructions that are sensitive
Using sensitive characters in user input without escaping them Impacts
The disclosure of sensitive data without authorization, data theft, or the abuse of data may result in a loss of income, reputation, and integrity for the company concerned.
Victims may be exposed to legal action in the future[13].

**5 .SQLIA DETECTION AND PREVENTION TECHNIQUES**

**5.1Machine Learning (ML) approach**

machine learning (ML) The part of In artificial intelligence that aims to give computers the capacity to learn from data and to predict the best model to apply.[1]
In this technique, the SQLIA (SQL Injection Attack) detection challenge Is modeled as a data analysis binary classification issue, with the SQLIA detection framework being constructed as the foundation. There are four main phases to this structure.[14]

During the data collection phase, It is necessary to generate two classes of queries for the normal and malicious classes. During the data preparation phase, it is necessary to transform the query into n_dimensional feature vectors. The feature extractor module gathers syntactic and semantic characteristics from the query, which are then fed back into the query. This is followed by the generation of multi-dimensional sequences. It is necessary to develop certain binary classification models during the training phase. Then, after evaluating the many models that have been built, the best binary classification model is selected. The evaluator model assesses the performance of the binary classification models based on a number of metrics, and the best model is then selected. During the detecting phase, the testing data is enhanced with fresh inputs to provide a complete picture. The data is predicted to have a class label assigned to it.
During the data preparation phase, a query is turned into a n_dimensional feature vector, which is used to store the testing data. With the help of the improved binary classification mode, the classifier module is able to distinguish between a malicious and a normal feature vector.



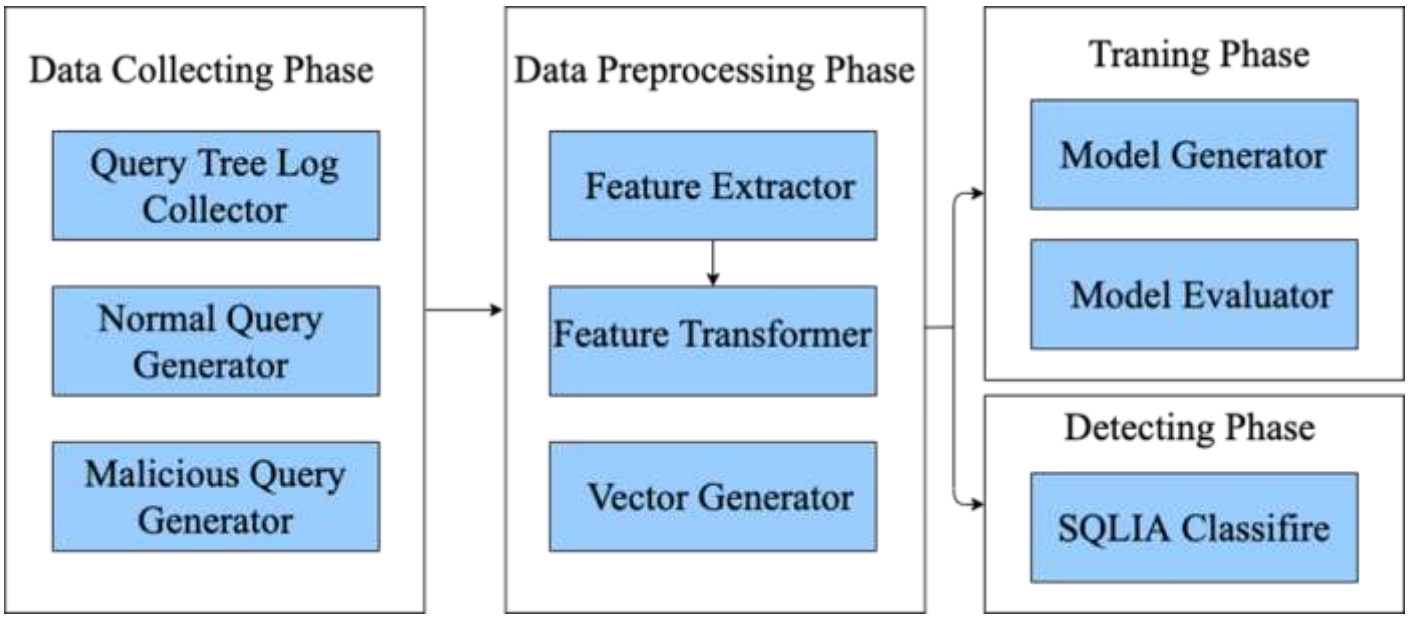


Fig. 2. SQLIA detection framework[14]

Several researchers depend on machine learning methods [15][16][17][18][19][20][21]–[24][25]. With machine learning, the suggested models are capable of learning without the need to express programming. A dataset that is

as close to reality as feasible in terms of order and structure guarantees that the classifier's training and testing phases are successful in terms of completion. Thus, it can detect and prevent SQLIA successfully shown Accuracy action in Table 1.

TABLE 1: Machine Learning Technique s(ML) for SQLIA Problem

| Techniques | Classifier(s) | Accuracy(A) / Precision(P) |
|---|---|---|
| HIPS-[15] | Bayesian-Naif<br>Bayesian-Multinomial | A= 97.6% |
| SQLI-IDS-[16] | (BNN) Backpropagation-Neural Network | A= 96.8% |
| Sheykhkanloo-[17] | (NN) Neural Network Model | A= 95% |
| Verbruggen - [18] | (DT)Decision Tree / SVM/Random Tree/ Jribber/ Neural Network/Random Forest | A=98.6% for (NN) |
| Wang - [19] | Stacked Auto-Encoder | P =100% |
| Ingre - [20] | Decision Tree(DT) | A= 83.7% |
| Moosa -[21] | Neural network Multilayer Feed Forward(FNN) | A= 66.67% |
| Joshi - [24] | Naïve-Bayes | A= 93.9% |
| SQLi_GOT- [22] | Super vector Machin( SVM) | A= 96.23% |
| Modsecurity - [23] | K-NN(K=3)/ SVM/ Random Forest | P=97% |
| TbD-NNbR-[25] | Neural Network (NN) | A = 99.23% |



**5.2 traditional approach(Non-Machine Learning (ML) approach)**

Following are some traditional approach for detecting and preventing SQL injection attacks.

**5.2.1 Static Analysis**

Static analysis is an approach that identifies and prevents vulnerabilities and malicious code from a system's source code before it is put through its paces during the execution stage [6]. SQLIA or other vulnerabilities may be detected and prevented using this approach, which is very often utilized by developers before the code is built and executed

**5.2.2 Dynamic Analysis**

During the process of the runtime, a model is created to identify SQLIA.Using this detection model, a query is executed before it is sent to the database server [26]. The outcome is that SQLIA is identified and blocked before the query reaches the database server. The overhead associated with building the model at runtime is one of the drawbacks of using this method[27].

**5.2.3 Combined Approach**

The Combined Approach makes use of the benefits of both static analysis and dynamic analysis in order to identify the detection and prevention of SQLIA [28]. During the static analysis, the area is identified, and then a model is generated that contains all of the valid queries that may be done in that area [9]. During execution, the queries are compared to the model that they are based on. If the queries do not correspond to the model, they will not be forwarded to the database for execution and will instead be rejected.

**5.2.4 Recommended Countermeasures**

Here are some measures that should be followed while developing a web-based application.

—Disable unused features: Disabling any and all superfluous features or functions is a good technique to avoid SQLIA since they may be potentially harmful.

—Custom error message: If the server receives malicious code or input, it may emit error messages as a response. In these error messages, there is information about the database that may be utilized to hack into the system.

In order to prevent it from revealing too much information, a custom error message is normally built up before it is executed.

—Escape functions: The escape functions protect the server against the majority of threats in a short amount of time.

—Prohibit certain keywords: There is a considerable likelihood that certain keywords such as UNION, DROP, and other malicious symbols may be used in an injection attack; hence they must be kept out of the system. The validation of user input is a critical component of SQLIA protection.

—Limit the size of the data: The amount of data that the user may enter should be restricted.
Some injections need the use of a specific amount of characters. Consequently, restricting the quantity of input data may avoid certain SQLIA.
—Use the Prepare Statements: A prepared statement is a SQL statement that is used to efficiently execute similar SQL queries. This is the most effective solution for safeguarding against SQL injections today. It will take some time, but it will be worth it, in the long run, to keep the server safe.
—Query Parameterization: Distinguish the SQL query from any type of parameters that could be sent along in a refund request to the database[6].
There are a number of other prevention techniques that have proven successful in the prevention of SQL injection, including AMNESIA, SQL Check, SQL Guard, and CANDID. On the other hand, SQL-DOM/ SQL_rand/AMNESIA/ Tainting/ SQL_Check/ SQL_Guard and CANDID have all proven unsuccessful in the prevention of a Stored Procedure Attack [29].
In Table 2, the char "S" refers to a technique that is capable of successfully stopping all typesof SQLIA, whereas the char "F" refers to a technique that failsto stop all typesof SQLIA, and the symbol "P" refers to a technique that only partially stops all type SQLIA due to limitations of the underlying approach, respectively. Due to the fact that only a handful of the detection and prevention approaches have been applied in practice[30], the findings shown in Table 2 are based on analytical evaluation rather than empirical evaluation.



TABLE 2- Comparative of Detection and Prevention Techniques with all typesof SQLIA

| Technique | Tautologies | Illegal or Logically incorrect query | Alternate Encodings | Union query | Piggyback Queries | Stored Procedure | Inference |
|---|---|---|---|---|---|---|---|
| AMNESIA[31] | S | S | S | S | S | F | S |
| [11] | S | S | S | S | S | S | S |
| SAFELI [32] | F | S | S | S | S | S | S |
| WASP [33] | S | S | S | S | S | S | S |
| R-WASP [34] | S | S | S | S | S | F | S |
| SecuriFly[35] | P | P | P | P | P | P | P |
| JDBC-Checker [36] | P | P | P | P | P | P | P |
| CANDID [37] | P | P | P | P | P | P | P |
| Swaddler[38] | P | P | P | P | P | P | P |
| DIWeDa[39] | F | F | F | F | F | F | S |
| Positive Tainting [40] | S | S | S | S | S | S | S |
| Automated approaches [41] | S | S | F | S | S | F | S |
| SQLIPA [42] | S | F | F | F | F | F | F |

Table 3 compares the various SQL injection suggested solutions available.[ Detection (D) / prevention (P) / report generation (RG) /modeling (M) / classification (C)/ Yes (✓) /No (✗)] were the skills that we included for each solution in the table 3. Furthermore, we list the SQLI attack types that the suggested solution is designed to stop.

The fact that certain solutions, such as [37][43], are based on the modeling of a dynamic SQL query and then comparing it with a collection of SQL query legal models is worth emphasizing.
However, although these techniques have the benefits of being simple and straightforward to apply, obtaining all of the legal query models is a challenging effort to do. It is possible that a lack of sufficient

lawful query models may result in a high false-positive rate, which will cause the web application to become inoperable. Other strategies, such as [44][45], are based on the randomness of the SQL command and fall into this group.The concept behind these techniques is to generate a random token for the SQL query. It is possible to accomplish this randomization by sequencing a random number or by including a cryptographic spice in the mix. Once the query is determined to be genuine, the randomization is turned off for that query. The most significant disadvantage of these strategies is that the randomization process has the potential to affect the intended user request while simultaneously restricting the range of available user input .

Techniques based on monitoring and auditing, such as [46][47], often attempt to audit and analyze code in order to identify potentially susceptible sections of code. Some techniques make prevention by repairing the code that is susceptible to attack. Others are just concerned with detecting the SQLI attack. They have the benefit of being able to operate without the need for a web connection prior to the deployment of the web application, allowing them to prevent and detect SQLIAs before the web application is deployed.

This strategy may result in a high proportion of false negatives since certain attacks may be initiated during runtime and, as a result, are undetected while operating in the not online mode.

A number of approaches make use of information theory, and they may identify the existence of SQLIA by going to measure the entropy of the query [48][49]. Hackers that are able to generate malicious queries with a sufficient level of entropy may circumvent these measures.

Ontology-based defenses such as [50][51] and [52] attempt to mitigate the vulnerabilities of signature-based approaches by providing excellent models of the SQLI attack types. Developing an ontology to model SQLIA is one of the most important challenges for the ontology; some studies attempt to describe the SQLIA, while others are enhanced with semantic rules to assist the ontology model in preventing or detecting SQLIA.

The primary benefit of using ontology as a detection approach is that it ensures the semantic relationships between the various components of the attack signature, which is a significant advantage.

Added to that Several researchers depend on machine learning methods [41, 43,44, 45, 48, 49, 51, 54, 59]. With machine learning, the suggested models are capable of learning without the need to express programming. A dataset that is as close to reality as feasible in terms of order and structure guarantees that the classifier's training and testing phases are successful in terms of completion. Thus, it is capable of detecting and preventing SQLIA successfully.



Table 3 compares the various SQL injection suggested solutions

| Technique | (D) Detection<br>(P) Prevention<br>(RG) Report Generate<br>(M) modeling<br>(C) Classification | Tautologies | Illegal or Logically incorrect query | Alternate Encodings | Union query | Piggyback Queries | Stored Procedure | Inference |
|---|---|---|---|---|---|---|---|---|
| CANDID-[37] | D/P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SQLStor-[43] | P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | | | | | | | |
| OBFUSCATION[44] | D | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Auto-Rand-[45] | D/P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | | | | | | | |
| PSIAQOP [46] | D/P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Joan Audit [47] | D/RG | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | | | | | | | |
| µ4SQLi [48] | D | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Das -et al-[49] | D | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | | | | | | | |
| Denker -et al.[50] | M | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Abderazed -et al. [51] | M/D/P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NSSA [52] | D/P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | | | | | | | |
| HIPS-[15] | C/D/P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SQLI-IDS-[16] | C/D | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Verbruggen - [18] | C/P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Wang - [19] | C/D | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Ingre - [20] | C/D | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Moosa -[21] | C/D | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Joshi - [24] | C/D | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SQLi_GOT- [22] | C/D | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Modsecurity - [23] | D/P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| TbD-NNbR-[25] | D/P | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |



## 6 .CONCLUSION

In light of the above discussion and research, it is clear that SQLIAs are among the most dangerous threats to applications and are connected to a relational database(SQL). In this study, the examined types of SQLIA, which is one of the most widely used attack types currently in use. In addition to the different types ofSQLIA , functioning techniques detection and prevention measures for all the different types of SQLIA. When evaluating detection and prevention strategies, compare their capacity to detect an attack and their ability to prevent an assault in terms of detection and prevention effectiveness. To overcome the SQLIA, it is necessary to increase the efficacy of several strategies used in the tables 3.

## REFERENCES

[1] D. Chen, Q. Yan, C. Wu, and J. Zhao, “Sql injection attack detection and prevention techniques using deep learning,” in *Journal of Physics: Conference Series*, 2021, vol. 1757, no. 1, p. 12055.

[2] “2015 Web Application Attack Report (WAAR),” *Application Defense Center (ADC)*, 2015. http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed%0A6.pdf .

[3] O. T. 10, “Top 10 2013-A1-Injection.” 2013, [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-A1-Injection .

[4] K. Ross, M. Moh, T.-S. Moh, and J. Yao, “Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection,” in *Proceedings of the ACMSE 2018 Conference*, 2018, pp. 1–8.

[5] N. Chaubey and S. Sharma, “A Discriminative Survey on SQL Injection Methods to Detect Vulnerabilities in Web applications,” vol. 7, no. 6, pp. 8–13, 2016.

[6] M. Shachi, N. S. Shourav, A. S. S. Ahmed, A. A. Brishty, and N. Sakib, “A Survey on Detection and Prevention of SQL and NoSQL Injection Attack on Server-side Applications,” *Int. J. Comput. Appl.*, vol. 183, no. 10, pp. 1–7, 2021, doi: 10.5120/ijca2021921396.

[7] J. Viegas and A. Orso, “A classification of SQL-injection attacks and countermeasures,” in *Int’l Symp. on Secure Software*, 2006, vol. 1, pp. 13–15, [Online]. Available: http://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf.

[8] A. Alazab, M. Alazab, J. Abawajy, and M. Hobbs, “Web application protection against SQL injection attack,” in *Proceedings of the 7th International Conference on Information Technology and Applications*, 2011, pp. 1–7.

[9] A. Ghafarian, “A hybrid method for detection and prevention of SQL injection attacks,” in *2017 Computing Conference*, 2017, pp. 833–838.

[10] K. Elshazly, Y. Fouad, M. Saleh, and A. Sewisy, “A Survey of SQL Injection Attack Detection and Prevention,” *J. Comput. Commun.*, vol. 02, no. 08, pp. 1–9, 2014, doi: 10.4236/jcc.2014.28001.

[11] A. Alazab and A. Khresiat, “New strategy for mitigating of SQL injection attack,” *Int. J. Comput.*

*Appl.*, vol. 154, no. 11, 2016.

[12] W. G. J. Halfond and A. Orso, "Detection and prevention of SQL injection attacks," in *Malware Detection*, Springer, 2007, pp. 85–109.

[13] S. M. H. Chaki and M. Mat Din, "A Survey on SQL Injection Prevention Methods," *Int. J. Innov. Comput.*, vol. 9, no. 1, pp. 47–54, 2019, doi: 10.11113/ijic.v9n1.224.

[14] M.-Y. Kim and D. H. Lee, "Data-mining based SQL injection attack detection using internal query trees," *Expert Syst. Appl.*, vol. 41, no. 11, pp. 5416–5430, 2014.

[15] A. Makiou, Y. Begriche, and A. Serhrouchni, "Improving Web Application Firewalls to detect advanced SQL injection attacks," in *2014 10th International Conference on Information Assurance and Security*, 2014, pp. 35–40.

[16] N. M. Sheykhkanloo, "SQL-IDS: evaluation of SQLi attack detection and classification based on machine learning techniques," in *Proceedings of the 8th International Conference on Security of Information and Networks*, 2015, pp. 258–266.

[17] N. M. Sheykhkanloo, "A learning-based neural network model for the detection and classification of SQL injection attacks," *Int. J. Cyber Warf. Terror.*, vol. 7, no. 2, pp. 16–41, 2017.

[18] R. Verbruggen and T. Heskes, "Creating firewall rules with machine learning techniques," *Nijmegen Netherlands Kerckhoffs Inst. Nijmegen*, pp. 9–27, 2014.

[19] Z. Wang, "The applications of deep learning on traffic identification," *BlackHat USA*, vol. 24, no. 11, pp. 1–10, 2015.

[20] B. Ingre, A. Yadav, and A. K. Soni, "Decision tree based intrusion detection system for NSL-KDD dataset," in *International conference on information and communication technology for intelligent systems*, 2017, pp. 207–218.

[21] A. Moosa, "Artificial neural network based web application firewall for SQL injection," *Int. J. Comput. Inf. Eng.*, vol. 4, no. 4, pp. 610–619, 2010.

[22] D. Kar, S. Panigrahi, and S. Sundararajan, "SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM," *Comput. Secur.*, vol. 60, pp. 206–225, 2016.

[23] G. Betarte, Á. Pardo, and R. Martínez, "Web application attacks detection using machine learning techniques," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018, pp. 1065–1072.

[24] A. Joshi and V. Geetha, "SQL Injection detection using machine learning," in *2014 international conference on control, instrumentation, communication and computational technologies (ICCICCT)*, 2014, pp. 1111–1115.

[25] T. K. George, K. P. Jacob, and R. K. James, "Token based detection and neural network based reconstruction framework against code injection vulnerabilities," *J. Inf. Secur. Appl.*, vol. 41, pp. 75–91, 2018.

[26] G. Shrivastava and K. Pathak, "SQL injection attacks: Technique and prevention mechanism," *Int. J. Comput. Appl.*, vol. 69, no. 07, pp. 975–8887, 2013.

[27] Website. . [Online], "Available: D. Box and A. Hejlsberg. LINQ: .NET Language-Integrated Query.," 2022. https://msdn.microsoft.com/en-us/library/bb308959.aspx.

[28] I. Lee, S. Jeong, S. Yeo, and J. Moon, "A novel method for SQL injection attack detection based on removing SQL query attribute values," *Math. Comput. Model.*, vol. 55, no. 1–2, pp. 58–68, 2012.

[29] H. Alsobhi and R. Alshareef, "SQL Injection Countermeasures Methods," in *2020 International Conference on Computing and Information Technology (ICCIT-1441)*, 2020, pp. 1–4.

[30] V. Nithya, R. Regan, and J. Vijayaraghavan, "A survey on SQL injection attacks, their detection and prevention techniques," *Int. J. Eng. Comput. Sci*, vol. 2, no. 4, pp. 886–905, 2013.

[31] W. G. J. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 795–798.

[32] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao, "A static analysis framework for detecting SQL injection vulnerabilities," in *31st annual international computer software and applications conference (COMPSAC 2007)*, 2007, vol. 1, pp. 87–96.

[33] W. Halfond, A. Orso, and P. Manolios, "Wasp: Protecting web applications using positive tainting and syntax-aware evaluation," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 65–81, 2008.

[34] M. Medhane, "R-WASP: Real time-web application SQL injection detector and preventer," *Int. J. Innov. Technol. Explor. Eng.*, vol. 2, no. 5, pp. 327–330, 2013.

[35] M. Martin, B. Livshits, and M. S. Lam, "Finding application errors and security flaws using PQL: a program query language," *Acm Sigplan Not.*, vol. 40, no. 10, pp. 365–383, 2005.

[36] C. Gould, Z. Su, and P. Devanbu, "JDBC checker: A static analysis tool for SQL/JDBC applications," in *Proceedings. 26th International Conference on Software Engineering*, 2004, pp. 697–698.

[37] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic candidate

evaluations for automatic prevention of SQL injection attacks," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, pp. 1–39, 2010.

[38] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swaddler: An approach for the anomaly-based detection of state violations in web applications," in *International Workshop on Recent Advances in Intrusion Detection*, 2007, pp. 63–86.

[39] A. Roichman and E. Gudes, "DIWeDa-detecting intrusions in web databases," in *IFIP Annual Conference on Data and Applications Security and Privacy*, 2008, pp. 313–329.

[40] W. G. J. Halfond, A. Orso, and P. Manolios, "Using positive tainting and syntax-aware evaluation to counter SQL injection attacks," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, 2006, pp. 175–185.

[41] M. Junjin, "An approach for SQL injection vulnerability detection," in *2009 Sixth International Conference on Information Technology: New Generations*, 2009, pp. 1411–1414.

[42] S. Ali, S. K. Shahzad, and H. Javed, "Sqlipa: An authentication mechanism against sql injection," *Eur. J. Sci. Res.*, vol. 38, no. 4, pp. 604–611, 2009.

[43] S. Mamadhan, T. Manesh, and V. Paul, "SQLStor: Blockage of stored procedure SQL injection attack using dynamic query structure validation," in *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)*, 2012, pp. 240–245.

[44] R. Halder and A. Cortesi, "Obfuscation-based analysis of SQL injection attacks," in *The IEEE symposium on Computers and Communications*, 2010, pp. 931–938.

[45] J. Perkins, J. Eikenberry, A. Coglio, D. Willenson, S. Sidiroglou-Douskos, and M. Rinard, "AutoRand: Automatic keyword randomization to prevent injection attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016, pp. 37–57.

[46] E. Al-Khashab, F. S. Al-Anzi, and A. A. Salman, "PSIAQOP: preventing SQL injection attacks based on query optimization process," in *Proceedings of the Second Kuwait Conference on e-Services and e-Systems*, 2011, pp. 1–8.

[47] J. Thomé, L. K. Shar, D. Bianculli, and L. C. Briand, "Joanaudit: A tool for auditing common injection vulnerabilities," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 1004–1008.

[48] D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan, "Automated testing for SQL injection vulnerabilities: an input mutation approach," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 259–269.

[49] D. Das, U. Sharma, and D. K. Bhattacharyya, "Defeating SQL injection attack in authentication security: an experimental study," *Int. J. Inf. Secur.*, vol. 18, no. 1, pp. 1–22, 2019.

[50] G. Denker, L. Kagal, and T. Finin, "Security in the Semantic Web using OWL," *Inf. Secur. Tech. Rep.*, vol. 10, no. 1, pp. 51–58, 2005.

[51] A. Razzaq, Z. Anwar, H. F. Ahmad, K. Latif, and F. Munir, "Ontology for attack detection: An intelligent approach to web application security," *Comput. Secur.*, vol. 45, pp. 124–146, 2014.

[52] G. Xu, Y. Cao, Y. Ren, X. Li, and Z. Feng, "Network security situation awareness based on semantic ontology and user-defined rules for Internet of Things," *IEEE Access*, vol. 5, pp. 21046–21056, 2017.