Attacks on SQL Injection and Developing Compressive Framework Using a Hybrid and Machine Learning Approach

Fitsum Gizachew Deriba (computer.fitsum@gmail.com)
Wachemo University Hossana
Tsegay Mullu Kassa
Wachemo University Hossana
Wubetu Barud Demilie
Wachemo University Hossana

Research Article

Keywords: SQL injection, machine learning, the security flaw

Posted Date: February 8th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1321852/v1

Additional Declarations: No competing interests reported.

Version of Record: A version of this preprint was published at Journal of Big Data on December 30th, 2022. See the published version at https://doi.org/10.1186/s40537-022-00678-0.

Abstract

Web applications play an important role in everyday life. Various Web applications are used to carry out billions of online transactions. These applications are vulnerable to attacks due to their extensive use. The most prevalent attack is SQL injection, which accepts user input and runs queries in the backend based on the user's input, returning desired results. To counter the SQL injection attack, various approaches have been offered; however, the majority of them either fail to cover the full breadth of the problem. This research paper looks into frequent SQL injection attack forms, their mechanisms, and a way of identifying them based on the SQL query's existence. We propose a comprehensive framework for determining the effectiveness of techniques that address certain issues following the essence of the attack, using hybrid (Statistic and dynamic) and machine learning. An extensive examination of the model based on a test set indicates that the Hybrid and ANN approaches outperform Naive Bayes, SVM, and Decision trees in terms of accuracy in classifying injected Queries. However, when it came to web loading time during testing, Nave Bayes outperformed. The Hybrid Method improved the accuracy of SQL injection attack prevention, according to the test findings. Although we used a limited dataset for training and testing in our study, it is advised that the dataset be expanded and the model be tested in a real-world setting.

1. OVERVIEW OF SQL QUERY ATTACKS

One of the major topics to be investigated in this study is web attacks. Even though there is a lot of web attack, SQL injection is one of the most common ones and top 5 web attack in 2021 according to the OWASP report. This attack gives attackers unrestricted access to databases that contain sensitive information [1][2]. To attack a web application, the attacker must first recognize and identify the system's weaknesses.

A web application comprises three levels, as is common knowledge: The First Presentation tier collects user feedback and displays the processing results to the user. The user is directly communicated via the presentation tier. Server script, the second control layer, processes data entered by the user and transmits the result to the database tier. The database tier returns the processed data to the Control tier, which is subsequently transmitted to the presentation tier for the user to access[3], [4],[5],[6] As a result, data processing in the web application occurs on the Control Stage, which can be implemented in a variety of server scripting languages. Finally, the DB stage saves and retrieves the data. All sensitive web application data is stored and managed in the database. Because this layer is directly connected to the Control tier and has no security checks, data in the database can be exposed and modified if the Control tier is successfully attacked. The general concept of web-based architecture is depicted in Figure 1.

The difficulty in perceiving the injected query at the database layer necessitates a system that controls and filters the query at the presentation layer[7] based on predetermined parameters. Various studies have been conducted to identify and prevent the injected Query. The bulk of them, however, do not identify all types of SQL injection, but they fared better on a handful or in the statistical or dynamic portions. Vulnerabilities in web apps can exist if the sanitization function does not correctly sanitize user input. Static analysis cannot tell whether or not input has been sanitized properly. Vulnerabilities often go unnoticed due to such flaws in static assessments.

SQL Prevent checks produced queries for those parameters and provide an alarm when an HTTP request parameter influences the syntax structure of a query[8]. User inputs are tracked using dynamic approaches, and a profile of caring queries is created. This technique suffers from false positives and false negatives[9] due to the inability to encapsulate input in the application that generates the query. As a result, we've discovered that the SQL injection attack forms still have a hole to fill.

The primary goal of this research is to examine current SQL injection attacks, identify their methodologies, strengths, weaknesses, and finally propose a thorough detection method. As a result of these SQL Injection attacks, there is a need and desire to develop a better SQL Injection detection system.

The main contributions to this work are:

- Studying and identifying the nature of SQL injection attacks and proposing prevention mechanisms.
- Proposing a comprehensive framework that detects all types of SQL attacks.

The remaining parts of this research are arranged as follows: section 2 offers basic information on various current SQL injection methods. Section 3 outlines the most SQL injection attack types and investigates them. Then, in section 4, methods for defining a developed compressive framework to SQL injection. Followed section 5 by result and discussion. Finally, section 6, will be a conclusion.

2. EXISTING METHODS FOR SQLI DETECTION AND PREVENTION

Several approaches have been proposed to detect and avoid SQL injection, some focusing on statistical analysis[10], [11],[12],[13],[14] or dynamic analysis[15][16], others on Hybrid approach[17][18]. These approaches are used for vulnerability analysis, scanning, detection and prevention, mitigation, and attack avoidance for web applications. There have been numerous studies to investigate vulnerability analysis[19] by a detailed examination of a web application's security holes. This has also been explored in prior studies by vulnerability scanning tools that are used in[20]. Many authors have recognized detection and prevention[21] are the best way to avoid attack from web application attacks. For instance, the following studies were conducted on the detection and prevention of SQL injection attacks;

[22] present all SQL injection attack types and also different tools which can detect or prevent these attacks. All types of SQL injection attacks and also different techniques which can detect or prevent them are presented. [23] present a comprehensive review of different types of SQL injection detection and prevention techniques. [24] propose a method for increasing the system's capability to detect and prevent SQL injection attacks based on the removal of SQL query attribute values and a honeypot for trapping attackers. [25]focus the types of SQL injection attacks on the open-source database in MySQL

This detection and prevention of attack were successfully established by various approaches including machine learning, hybrid, and web-based tools.

2.1. Machine Learning approach

Several types of research implied that machine learning approaches [26]–[29] can be employed to develop vulnerability predictors. The goal, regardless of the technique used, is to learn data associated with injection, which can then be used to predict vulnerability for new injection. A vulnerability analysis method needs to be able to adapt when more advanced security threats are discovered. Machine learning allows for re-training to respond to new vulnerability trends.

2.2. Hybrid (Statistic and Dynamic)

The majority of prior research has applied in a hybrid approach[8], [18], [30]. This can be done by comparing the structure of the queries to detect attacks. Initially, it detected if a dynamically generated query[31] has a different structure or grammar that meets certain requirements like data length, range, and form by Input validation and input purification by allowing only predefined characters and refusing all others, including those with unique significance to the interpreter than a static query[32] were followed. A new approach is therefore needed for SQLI attacks.

2.3. Developing a web-based framework

Many frameworks [33]–[35] have been developed and tested with various parameters. This has been discussed by a great number of authors in literature.

[36] propose a framework based on misuse and anomaly detection techniques to detect SQL injection attacks. [37] discuss a secure mechanism for protecting web applications from SQL Injection attacks by using framework and database firewall. [32] present a framework that can be used to handle tautology-based SQL Injection Attacks using the post-deployment monitoring technique. [38] evaluate Runtime Monitoring Framework to detect and prevent SQL Injection Attacks on web applications. [39] present a cloud computing adoption framework (CCAF) security suitable for business clouds. [40] propose SQL injection intrusion detection framework as a service for SaaS providers, SQLI IDaaS, which allows a SaaS provider to detect SQLIAs targeting several SaaS applications without reading, analyzing, or modifying the source code. To raise the tenants' awareness of the seriousness of SQLIAs. [34]introduce a novel traffic-based SQLIA detection and vulnerability analysis framework named DIAVA, which can proactively send warnings to tenants promptly.

On given parameters, some of them perform well, while others do not. As a consequence, these frameworks detect the injected queries but they have no control over them. A closer look at the literature on the above SQL injection attack, however, reveals many gaps and shortcomings.

3. MOST COMMON ATTACKS ON SQL INJECTION

To address the above-mentioned issue, we present a detailed overview of the various types of SQL injection attacks that have been discovered to date. We include explanations and examples of how attacks of that sort may be carried out, as well as explicit mitigation mechanisms, for each type of attack. Finally, we propose a comprehensive framework that avoids all forms of attacks.

3.1 Tautology attack

In this attack, the attacker tries to use a conditional question argument to test always true in the tautology attack. The attacker injects the condition and transforms it into a tautology that is always valid using the WHERE clause[41] The most common type of tautology attack, the nature of the attack, and the approach used to detect them is described below in table 1.

Table 1: Common tautology attacks

Type of injection	*Nature of attack*	*Approach for Detection*
String SQL injection	Bypassing Authentication, identifying injectable parameters using string data type, extracting data	Rule-based
Numeric SQL injection	Bypassing Authentication, identifying injectable parameters using numeric data type extracting data	Rule-based
Comment attack	Bypassing Authentication, identifying injectable parameters using the comment form, extracting data	Rule-based

The SQL query results convert the original condition into a tautology, allowing an unauthorized user to access all records in the database table, for example. Guardium detects many variants on tautological statements in database requests and prevents this form of attack. Previous studies were limited to investigating the common tautology attack but still, it needs to study all forms of the tautology attack that leads to injectable. Due to this, the study investigates the types of attacks and recommends an approach that is appropriate for each attack

3.2 Union Query

In this category of attack, the UNION operator [29] is only used if both queries have the same form, the attacker constructs a SELECT statement that is similar to the original query. To do this, you will need to know the correct table name, as well as the number of columns and their data types in the first query. As a result, two conditions might meet or an attack on the Union query will be launched, and each query returns the same number of columns[42]. If a column's data type is incompatible with string data, the injected query will result in a database error. Based on the nature of the attack the suggested approach is described below in table 2.

Table 2: Union Injection Attack

Type of injection	*Nature of attack*	*Recommended approach*
Union Query attack	Bypassing Authentication, extracting data using Union operation	Rule-based

Mostly the second query in a union is malicious [43], and for instance the text after (--) is ignored since it acts as a comment for the SQL Parser. Taking advantage of this, the attacker uses this query to target the online application or website.

3.3 Piggybacked Query

Data extraction, data addition or modification, denial of service, and remote command execution are all examples of attack determined. An attacker tries to inject additional queries into the original query in this type of attack. This form differs from others in that attackers are attempting to add new and distinct queries that "piggyback" on the original query rather than changing it[44][45]. As a result, several SQL queries are sent to the database. Table 3 states the nature and appropriate approach used for attack.

Table 3: Piggy Backed Query

Type of Injection	*Nature of attack*	*Recommended approach*
Piggybacked query	Adding or altering data, performing denial of service, and executing remote commands are all examples of data extraction.	Machine Learning

These kinds of criminal behaviors can be prevented by first finding the right SQL Query via adequate validation or by employing various detection mechanisms. Static Analysis can avoid this form of attack; run-time monitoring is not required.

3.4 Illegal/incorrect Query

Identifying injectable parameters, performing database finger-printing, and extracting data are all part of the attack purpose. This attack helps an attacker to collect crucial information[37] about the type and function of a Web application's back-end database. The attack is thought to be a warm-up for future attacks, gathering information. This attack takes advantage of the fact that application servers' default error pages[46] are frequently excessively descriptive. Due to that the recommended approach is shown in table 4.

Table 4: Illegal or Incorrect Query

Type of Injection	*Nature of attack*	*Recommended approach*
Illegal/ incorrect Query	Error messages ignored by the client are used to locate useful data, allowing the backend database to be injected more easily.	Machine Learning

In general, this attack takes advantage of the error message produced by the database when a query is wrong.

3.5 Stored Procedure Query

The users can save their features and use them whenever they want. A collection of SQL queries is provided with the feature to use it. The intruder uses malicious SQL Injection codes to execute the database's built-in stored procedures[47]. This leads to cause the cached stored procedure query plans being recompiled. A Stored Procedure's constraint is that it can only be used in the database. The best method to overcome this attack is described below in table 5.

Table 5: Stored Procedure Query

Type of Injection	*Nature of attack*	*Recommended approach*
Stored procedure query	Performing privilege escalation, denial of service, and remote command execution.	Rule-based

3.6 Inference Query

The query is recast as an operation in this attack, and it is executed based on the response to a true/false question about database data values[48]. For this method of injection, attackers attempt to break into a site that has been sufficiently protected that when an injection is successful, there is no accessible feedback in the form of database error messages. Since database error messages are not available to provide feedback, attackers must rely on another approach to get a response from the database. Different forms of attack under inference query are shown in table 6 below.

Table 6: Common inference Query attack

Types of attack	*Nature of attack*	*Recommended approach*
Blind SQL injection[41]	Collect valuable data by inferring from the page's answers after asking the server a set of true/false questions.	Machine Learning
Timing Attack[45]	Observe the response time, which will assist the attacker in making an informed decision about which injection approach to use.	Machine Learning
Database backdoor attack	Set a trigger to collect the user's feedback and send it to his or her e-mail address.	Machine Learning
Command SQL injection	Injecting and executing system-level commands via a vulnerable program is the essence of the attack.	Rule-based

3.7 Alternative Encoding Query

The injected text is changed in this attack to avoid detection by protective coding practices as well as several automated prevention

techniques. This form of attack is used in combination with others[49].To intend their attack they use the regular expression[21]. This implies they do not offer a special way to target an application; rather, they are an enabling technology that enables attackers to bypass detection and prevention strategies and exploit vulnerabilities which are described in table 7.

Table 7: Alternative encoding Query

Type of Injection	*Nature of attack*	*Recommended approach*
Alternative Encoding Query	Safe protective coding and automatic prevention systems are used to keep it from being detected.	Machine Learning

Due to all the above nature of tack and needed prevention and detection mechanism, Therefore, A more systematic and theoretical analysis is required for the Injection attacks.

4. PROPOSED FRAMEWORKS

To develop our framework, we studied the existing approaches with their attacking methods and their holes. Therefore, we propose a comprehensive framework that encompasses solving all vulnerabilities that exist in previous works. In proposed framework

The attacker must first open his browser to carry out the activity or attack task. Initially, the intruder either enters his password into the application or requests authorization to access the web service via the internet if the application is open. To get past the firewall checker, the intruder must first get past the firewall checker.

Then web server accepts user input by various mechanisms such as user input validation and then uses the input to create queries to an underlying database[8], [42]. This can be accomplished by identifying injection parameters, determining the type and version of the database used by a Web application, and determining database schema. If permission was granted based on the request, the attacker will request application server access again. However, in this situation, we suggested a model for evaluating whether or not the requested access involves SQL injection which is shown below in figure 2.

Before the classification of SQL queries, there were several stages. The first feature extraction is performed by using static and dynamic analysis to check whether the requested queries are injected with either approach. The classifier accepts queries and matches them with the trained data set based on the requested query. Then the machine learning classifier accepts the extracted feature and trains the model to identify the injected query. SVM[50], [51], Decision tree, Nave Bayes[18], [52], [53], and other algorithms in machine learning approaches [20], [54]–[58] are used to solve classification algorithm. The trained model passes all stages such as pre-processing and feature extraction.

So, the classifiers will train to recognize different types of SQL injection attacks according to the given trained machine learning model and hybrid approach at feature extraction steps. The model matches the pattern of each line query requested based on the trained pre-fetched and trained dataset. If the SQL query is injected with one form of qualified attack, the model will either reject the request or submit it to the application server and database server to access the requested operation if the query is pure SQL or no injection. As a result, we suggest creating a new architecture based on a hybrid and machine learning approach to obtain the best results possible when dealing with SQL query injection attacks.

5. RESULT AND DISCUSSION

Key insights come from the above brief review to analyze and assess previous studies' methodologies. We employed three injection parameters in various forms in this study. The first is through a user input field, which allows a web application used to request information from a backend database using HTTP POST and GET, and the second is through cookies, which may be used to restore a client's state information when they return to a Web application. If a Web application uses the contents of cookies to construct SQL queries, an attacker can exploit this vulnerability to change cookies and submit them to the database server. Finally, by analyzing session usage information and recognizing browsing behaviors, a server variable can be created. Because attackers can forge the values that are placed in HTTP and network headers by entering malicious input into the client-end of the application or by crafting their request to the server, if these variables are logged to a database without sanitization, this could result in SQL injection vulnerability. This attack parameter includes all of the attack types mentioned in this study.

All attacks sent to the server are logged and kept in the database as attack log data. In addition, the attack log data is separated into two categories: attacks and normal data. We collected around 11,847 thousand weblogs, cookies, and session usages to train and test the model. There are around 7621 thousand SQL normal logs and 4226 thousand injection attack logs among them. All logs are URL access

records with query statements for user fields. The dataset will be utilized as training data for the Naive Bayes, Decision Tree, SVM, and Hybrid methods to avoid SQL injection attacks. Before being utilized for classification, the dataset is trained. Python3 is the programming language and the experimental equipment is a laptop computer. The Keras framework is used, which is based on TensorFlow. The results of classifiers during the training phase are shown in Table 8.

Table 8: Result of different approaches on the training set

Approaches	Training set accuracy	Training time
Naïve Bayes	0.87301	5.2 sec
Decision tree	0.9526	47.8 sec
SVM	0.9843	16.5 sec
ANN	0.9916	2453 sec
Hybrid	0.9960	2609 sec

The simple understanding of ANN is to train several almost unrelated weak learners, according to the study SVM, Decision Tree, and Naïve Bayes is the based learners. The accuracy rate of the ANN training set is 99.16 percent, and the hybrid approach has a rate of 99.60 percent, making it the best trainer among others. The training time for Nave Bayes and SVM, on the other hand, is extremely short.

Table 9: Result of different approaches on the test set

Approaches	Test set accuracy	Testing time
Naïve Bayes	0.872	0.38 ms
Decision tree	0.943	0.67 ms
SVM	0.9681	0.96 ms
ANN	0.9852	3.11 ms
Hybrid	0.9927	5.49 ms

In above table 9, Hybrid and ANN outperform SVM, Decision Tree, and Naive Bayes on the test set. In contrast to the other two approaches, Nave Bayes, Decision tree, and SVM are better in terms of time used during testing. When it comes to detecting SQLI, ANN has consistently outperformed other machine learning algorithms.

6. CONCLUSION

In this paper, we evaluated different strategies for detecting and preventing SQL Injection. To begin with, we defined the various forms of SQL injection that have been discovered to date. After that, we assessed the techniques under consideration in terms of their ability to detect and prevent SQL attacks. We also investigated the various mechanisms and decided which techniques were capable of dealing with which mechanisms. Then we identify each technique's specifications and develop a comprehensive framework to detect and prevent SQL injection attacks using hybrid and machine learning techniques. Based on our model evaluation, we found that the hybrid approach and ANN are the best approaches to classify SQL injection. In our study, we used a small dataset for training and testing, but maximizing the dataset and performing the model in a real environment is recommended.

7. DECLARATIONS

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Availability of data and materials

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Funding

Not applicable.

Authors' contributions

Fitsum Gizachew Deriba and Tsegay Mullu Kassa: Have prepared the manuscript including analysis, data curation, visualization, conceptualization, methodology, writing of the original draft. Wubetu Barud Demilie: Performs the tasks including conceptualization, validation, writing, review, and editing the final work. All authors have read and approved the final manuscript.

Acknowledgments

Not applicable.

References

[1] M. Amin *et al.*, “Review of SQL Injection : Problems and Prevention,” vol. 2, pp. 215–219.

[2] A. Kumar and S. Binu, “Proposed Method for SQL Injection Detection and its Prevention,” vol. 7, pp. 213–216, 2018.

[3] G. Hendita and A. Kusuma, “Analysis of SQL Injection Attacks on Website Service,” vol. 1, no. 1, 2018.

[4] O. C. Abikoye, A. Abubakar, A. H. Dokoro, and O. N. Akande, “A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm,” 2020.

[5] X. Yun and W. Wen, *Cyber Security*. 2018.

[6] A. Alazab, “New Strategy for Mitigating of SQL Injection Attack,” vol. 154, no. 11, pp. 1–10, 2016.

[7] A. Gurina and V. Eliseev, “Anomaly-Based Method for Detecting Multiple Classes of Network Attacks,” pp. 1–24, 2019, DOI: 10.3390/info10030084.

[8] R. Jahanshahi, A. Doupé, and M. Egele, “You shall not pass : Mitigating SQL Injection Attacks on Legacy Web Applications,” pp. 445–457, 2020.

[9] I. Medeiros, M. Beatriz, N. Neves, and M. Correia, “SEPTIC: Detecting Injection Attacks and Vulnerabilities Inside the DBMS,” *IEEE Trans. Reliab.*, vol. 68, no. 3, pp. 1168–1188, 2019, DOI: 10.1109/tr.2019.2900007.

[10] M. K. Gupta, M. C. Govil, and G. Singh, “Static analysis approaches to detect SQL injection and cross-site scripting vulnerabilities in web applications: A survey,” *Int. Conf. Recent Adv. Innov. Eng. ICRAIE 2014*, pp. 9–13, 2014, DOI: 10.1109/ICRAIE.2014.6909173.

[11] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao, “A static analysis framework for detecting SQL injection vulnerabilities,” *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 1, no. Compsac, pp. 87–94, 2007, doi: 10.1109/COMPSAC.2007.43.

[12] M. Alenezi and Y. Javed, “Open source web application security: A static analysis approach,” *Proc. - 2016 Int. Conf. Eng. MIS, ICEMIS 2016*, 2016, doi: 10.1109/ICEMIS.2016.7745369.

[13] F. Spoto *et al.*, “Static Identification of Injection Attacks in Java,” vol. 41, no. 3, 2019.

[14] bhayakumara S. Basutakara and D. J. P N, “A Review of Static Code Analysis Methods for Detecting Security Flaws,” *J. Univ. Shanghai Sci. Technol.*, vol. 23, no. 06, pp. 647–653, 2021, DOI: 10.51201/jusst/21/05320.

[15] D. Das, U. Sharma, and D. Bhattacharyya, “An Approach to Detection of SQL Injection Attack Based on Dynamic Query Matching,” *Int. J. Comput. …*, vol. 1, no. 25, pp. 28–34, 2010, [Online]. Available: https://pdfs.semanticscholar.org/7691/0e9141d278ffddff7b7d8dd8ca53defb51cb.pdf%5Cnhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.7553&rep=rep1&type=pdf.

[16] S. Nanda, L. C. Lam, and T. C. Chiueh, “Dynamic multi-process information flow tracking for web application security,” *Proc. 8th ACM/IFIP/USENIX Int. Conf. Middlew. 2007, Middleware'07*, pp. 1–20, 2008, DOI: 10.1145/1377943.1377956.

[17] A. Makiou, Y. Begriche, and A. Serhrouchni, “Hybrid approach to detect SQLi attacks and evasion techniques,” *Collab. 2014 - Proc. 10th IEEE Int. Conf. Collab. Comput. Networking, Appl. Work.*, pp. 452–456, 2015, DOI: 10.4108/icst.collaboratecom.2014.257568.

[18] F. Y. Hernawan, I. Hidayatulloh, and I. F. Adam, “Hybrid method integrating SQL-IF and Naïve Bayes for SQL injection attack avoidance,” vol. 1, no. 2, pp. 85–96, 2020.

[19] S. P. K and A. Murugan, “Analysis of Vulnerability Detection Tool for Web Services,” vol. 7, pp. 773–778, 2018.

[20] P. Techniques *et al.*, “Design and Implementation of SQL Injection Vulnerability Scanning Tool Design and Implementation of SQL Injection Vulnerability Scanning Tool,” 2020, DOI: 10.1088/1742-6596/1575/1/012094.

[21] B. J. S. Kumar and P. P. Anaswara, “Vulnerability detection and prevention of SQL injection,” vol. 7, pp. 16–18, 2018.

[22] A. Tajpour, M. Massrum, and M. Z. Heydari, “Comparison of SQL injection detection and prevention techniques,” *ICETC 2010 - 2010 2nd Int. Conf. Educ. Technol. Comput.*, vol. 5, pp. 174–179, 2010, DOI: 10.1109/ICETC.2010.5529788.

[23] A. Sadeghian, M. Zamani, and A. A. Manaf, “A taxonomy of SQL injection detection and prevention techniques,” *Proc. - 2013 Int. Conf. Informatics Creat. Multimedia, ICICM 2013*, pp. 53–56, 2013, doi: 10.1109/ICICM.2013.18.

[24] S. Djanali, F. X. Arunanto, B. A. Pratomo, A. Baihaqi, H. Studiawan, and A. Mazharuddin, “Aggressive Web Application Honeypot for Exposing Attacker s Identity,” no. November 2014, DOI: 10.1109/ICITACEE.2014.7065744.

[25] W. G. J. Halfond and A. Orso, “Detection and Prevention of SQL Injection Attacks,” *Adv. Inf. Secur.*, vol. 27, no. 7, pp. 85–109, 2007, DOI: 10.1007/978-0-387-44599-1_5.

[26] T. Pattewar, H. Patil, H. Patil, N. Patil, M. Taneja, and T. Wadile, “Detection of SQL Injection using Machine Learning : A Survey,” pp. 239–246, 2019.

[27] M. Zolanvari, S. Member, M. A. Teixeira, S. Member, L. Gupta, and S. Member, “Machine Learning Based Network Vulnerability Analysis of Industrial Internet of Things,” pp. 1–14.

[28] M. A. Azman, M. F. Marhusin, R. Sulaiman, U. Sains, M. F. Marhusin, and U. Sains, “Machine Learning-Based Technique to Detect SQL Injection Attack,” pp. 1–8, 2021, DOI: 10.3844/jcssp.2021.296.303.

[29] S. S. A. Krishnan, A. N. Sabu, P. P. Sajan, and A. L. Sreedeep, “SQL Injection Detection Using Machine Learning,” vol. 11, no. 3, pp. 300–310.

[30] B. J. S. Kumar and K. Pujitha, “Web Application Vulnerability Detection Using Hybrid String Matching Algorithm,” vol. 7, pp. 106–109, 2018.

[31] S. Son, K. S. McKinley, and V. Shmatikov, “Diglossia: Detecting code injection attacks with precision and efficiency,” *Proc. ACM Conf. Comput. Commun. Secur.*, no. 2, pp. 1181–1191, 2013, DOI: 10.1145/2508859.2516696.

[32] R. Dharam and S. G. Shiva, “Runtime monitors for tautology based SQL injection attacks,” *Proc. 2012 Int. Conf. Cyber Secur. Cyber Warf. Digit. Forensic, CyberSec 2012*, pp. 253–258, 2012, DOI: 10.1109/CyberSec.2012.6246104.

[33] D. Y. Kao, C. J. Lai, and C. W. Su, “A Framework for SQL Injection Investigations: Detection, Investigation, and Forensics,” *Proc. - 2018 IEEE Int. Conf. Syst. Man, Cybern. SMC 2018*, no. 1, pp. 2838–2843, 2019, DOI: 10.1109/SMC.2018.00483.

[34] H. Gu *et al.*, “DIAVA: A Traffic-Based Framework for Detection of SQL Injection Attacks and Vulnerability Analysis of Leaked Data,” *IEEE Trans. Reliab.*, vol. 69, no. 1, pp. 188–202, 2020, DOI: 10.1109/TR.2019.2925415.

[35] W. C. Chung, H. P. Lin, S. C. Chen, M. F. Jiang, and Y. C. Chung, “JackHare: a framework for SQL to NoSQL translation using MapReduce,” *Autom. Softw. Eng.*, vol. 21, no. 4, pp. 489–508, 2014, DOI: 10.1007/s10515-013-0135-x.

[36] S. Ezzat, M. I., L. M., and Y. K., “Web Anomaly Misuse Intrusion Detection Framework for SQL Injection Detection,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 3, pp. 123–129, 2012, DOI: 10.14569/ijacsa.2012.030321.

[37] Y. V. N. Manikanta, “Protecting Web Applications from SQL Injection Attacks,” pp. 609–613, 2012.

[38] R. Dharam and S. G. Shiva, “Runtime Monitoring Framework for SQL Injection Attacks,” vol. 6, no. 5, 2014, DOI: 10.7763/IJET.2014.V6.731.

[39] V. Chang, Y. H. Kuo, and M. Ramachandran, “Cloud computing adoption framework: A security framework for business clouds,” *Futur. Gener. Comput. Syst.*, vol. 57, pp. 24–41, 2016, DOI: 10.1016/j.future.2015.09.031.

[40] M. Yassin, H. Ould-Slimane, C. Talhi, and H. Boucheneb, “SQLIIDaaS: A SQL Injection Intrusion Detection Framework as a Service for SaaS Providers,” *Proc. - 4th IEEE Int. Conf. Cyber Secur. Cloud Comput. CSCloud 2017 3rd IEEE Int. Conf. Scalable Smart Cloud, SSC 2017*, pp. 163–170, 2017, DOI: 10.1109/CSCloud.2017.27.

[41] G. Yiğit and M. Arnavutoğlu, “SQL Injection Attacks Detection & Prevention Techniques,” vol. 9, no. 5, 2017, DOI: 10.7763/IJCTE.2017.V9.1165.

[42] L. Erdődi, Å. Å. Sommervoll, and F. M. Zennaro, “Journal of Information Security and Applications Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents,” *J. Inf. Secur. Appl.*, vol. 61, no. July, p. 102903, 2021, DOI: 10.1016/j.jisa.2021.102903.

[43] “An Improved SQL Injection Attack Detection Model Using Machine Learning Techniques,” vol. 11, no. 1, pp. 53–57, 2021.

[44] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu, “DAPASA: Detecting Android Piggybacked Apps Through Sensitive Subgraph Analysis,” *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 8, pp. 1772–1785, 2017, DOI: 10.1109/TIFS.2017.2687880.

[45] B. Shunmugapriya and B. Paramasivan, “Protection Against SQL Injection Attack in Cloud Computing,” vol. 9, no. 02, pp. 502–510, 2020.

[46] K. Varshney and R. L. Ujjwal, “LsSQLIDP : Literature survey on SQL injection detection and prevention techniques,” *J. Stat. Manag. Syst.*, vol. 22, no. 2, pp. 257–269, 2019, DOI: 10.1080/09720510.2019.1580904.

[47] K. Ahmad and M. Karim, “A Method to Prevent SQL Injection Attack using an Improved Parameterized Stored Procedure,” vol. 12, no. 6, pp. 324–332, 2021.

[48] M. Kareem, “Prevention of SQL Injection Attacks using AWS WAF,” p. 47, 2018, [Online]. Available: http://repository.stcloudstate.edu/cgi/viewcontent.cgi?article=1094&context=msia_etds.

[49] S. Mohammed, H. Chaki, and M. M. Din, “A Survey on SQL Injection Prevention Methods,” vol. 9, no. 1, pp. 47–54, 2019.

[50] R. Rawat, “SQL injection attack Detection using SVM,” no. March 2012, 2020, DOI: 10.5120/5749-7043.

[51] Z. Chen and M. Guo, “Research on SQL injection detection technology based on SVM,” vol. 01004, pp. 1–5, 2018.

[52] A. Banchhor and T. Vaidya, “SQL INJECTION DETECTION USING BAYE ’ S CLASSIFICATION,” pp. 313–317.

[53] M. Olalere *et al.*, “A Naïve Bayes Based Pattern Recognition Model for Detection and Categorization of Structured Query Language Injection Attack,” vol. 7, no. 2, pp. 189–199, 2018.

[54] M. Liu and T. Chen, “DeepSQLi : Deep Semantic Learning for Testing SQL Injection,” pp. 286–297.

[55] T. Liu, Y. Qi, L. Shi, and J. Yan, “Locate-Then-Detect : Real-time Web Attack Detection via Attention-based Deep Neural Networks,” pp. 4725–4731, 2016.

[56] M. Volkova, P. Chmelar, and L. Sobotka, “MACHINE LEARNING BLUNTS THE NEEDLE OF ADVANCED SQL INJECTIONS,” vol. 25, no. 1, pp. 23–30, 2019.

[57] X. I. N. Xie, C. Ren, Y. Fu, J. I. E. Xu, and J. Guo, “SQL Injection Detection for Web Applications Based on Elastic-Pooling CNN,” *IEEE Access*, vol. 7, pp. 151475–151481, 2019, DOI: 10.1109/ACCESS.2019.2947527.

[58] Q. I. Li, W. Li, and J. Wang, “A SQL Injection Detection Method Based on Adaptive Deep Forest,” pp. 145385–145394, 2019, DOI: 10.1109/ACCESS.2019.2944951.

Figures

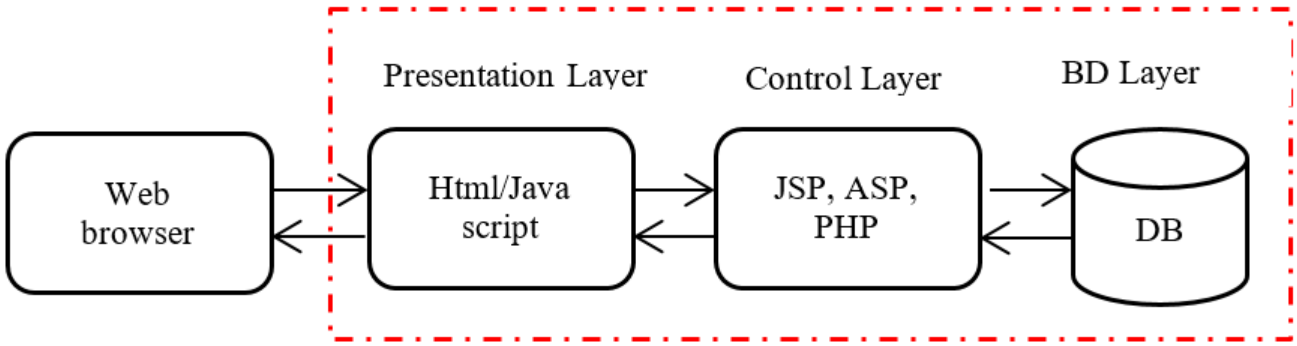

Figure 1

Web application Architecture

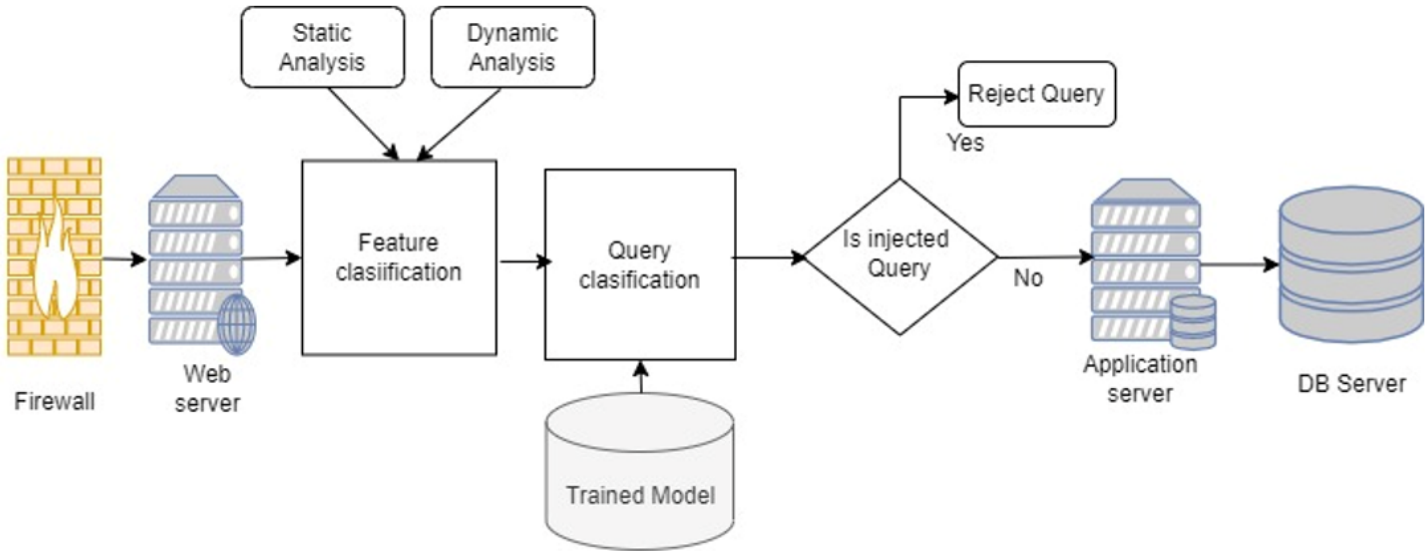

Figure 2

Proposed Frame work for SQL injection Detection