Fitsum Gizachew Deriba[1], Ayodeji Olalekan SALAU [2,*], Shaimaa Hadi Mohammed[3], Tsegay Mullu Kassa [4], Wubetu Barud Demilie [5]

Department of Computer Science, Wachemo University Hossana, Ethiopia (1),
Department of Electrical/Electronics and Computer Engineering, Afe Babalola University Ado-Ekiti, Nigeria (2),
Department of Computer Science, Summer University, Iraq (3)
Department of Information Technology, Wachemo University, Hossana, Ethiopia (4), (5)

doi:10.15199/48.2022.07.30

Development of a Compressive Framework Using Machine Learning Approaches for SQL Injection Attacks

Abstract. *Web applications play an important role in our daily lives. Various Web applications are used to carry out billions of online transactions. Because of their widespread use, these applications are vulnerable to attacks. SQL injection is the most common attack, which accepts user input and runs queries in the backend and returns the desired results. Various approaches have been proposed to counter the SQL injection attack; however, the majority of them have most times failed to cover the entire scope of the problem. This research paper investigates the frequent SQL injection attack forms, their mechanisms, and a way of identifying them based on the SQL query's existence. In addition, we propose a comprehensive framework to determine the effectiveness of the proposed techniques in addressing a number of issues depending on the type of the attack, by using a hybrid (Statistic and dynamic) approach and machine learning. An extensive examination of the model based on a test set indicates that the Hybrid approach and ANN outperforms Naive Bayes, SVM, and Decision tree in terms of accuracy of classifying injected queries. However, with respect to web loading time during testing, Naive Bayes outperforms the other approaches. The proposed Method improved the accuracy of SQL injection attack prevention, according to the test findings.*

Streszczenie. *Aplikacje internetowe odgrywają ważną rolę w naszym codziennym życiu. Różne aplikacje internetowe służą do przeprowadzania miliardów transakcji online. Ze względu na ich szerokie zastosowanie aplikacje te są podatne na ataki. Wstrzyknięcie SQL jest najczęstszym atakiem, który akceptuje dane wejściowe użytkownika i uruchamia zapytania w zapleczu oraz zwraca pożądane wyniki. Zaproponowano różne podejścia do przeciwdziałania atakowi SQL injection; jednak większość z nich przez większość czasu nie obejmowała całego zakresu problemu. W tym artykule badawczym przeanalizowano częste formy ataków typu SQL injection, ich mechanizmy oraz sposób ich identyfikacji na podstawie istnienia zapytania SQL. Ponadto proponujemy kompleksowe ramy do określania skuteczności technik, które rozwiązują określone problemy w zależności od rodzaju ataku, z wykorzystaniem podejścia hybrydowego (statystycznego i dynamicznego) oraz uczenia maszynowego. Obszerne badanie modelu na podstawie zestawu testowego wskazuje, że podejście hybrydowe i SNN przewyższają Naive Bayes, SVM i drzewo decyzyjne pod względem dokładności klasyfikacji wstrzykiwanych zapytań. Jednak pod względem czasu ładowania sieci podczas testowania, Naive Bayes przewyższa inne podejścia. Zgodnie z wynikami testów, zaproponowana metoda poprawiła dokładność zapobiegania atakom typu SQL injection.* (***Opracowanie spójnego systemua z wykorzystaniem metod uczenia maszynowego w atakach typu SQL Injection***)

Keywords: SQL injection, Machine Learning, Security flaw.
Słowa kluczowe: Wstrzyknięcie SQL, uczenie maszynowe, luka w zabezpieczeniach

Introduction

Web attacks are one of the most important topics to research in network security. Despite the fact that there are numerous web attacks, SQL injection is one of the most common and is predicted to be one of the top five web attacks in 2021, according to the OWASP report [1]. This attack gives attackers unrestricted access to databases that contain sensitive information [2]. To attack a web application, the attacker must first recognize and identify the system's weaknesses. A web application comprises of three levels, such as: The First is presentation tier which collects user feedback and displays the processing results of the user. The user is directly communicated via the presentation tier. The second control layer, server script, processes data entered by the user and sends the results to the database tier. The database tier sends the processed data to the Control tier, which then sends it to the presentation tier for the user to view [3-6]. 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 Database (DB) stage saves and retrieves the data. All sensitive web application data are 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 Queries.

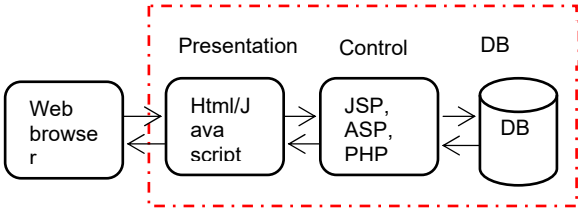


Fig. 1: Web application architecture.

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 the inputs have been sanitized properly. Vulnerabilities often go unnoticed due to such flaws in static assessments. SQL prevents such parameter checks on generated queries and gives an alert when an HTTP request parameter influences the syntax structure of the query [8]. User inputs are tracked using dynamic approaches, and a profile to check queries is created. This technique suffers from false positives and false negatives due to its inability to encapsulate input in the application that generates the query [9]. As a result, we've discovered that the SQL injection attack forms still have a number of limitations.The primary goal of this research study is to examine current SQL injection attacks, identify their methodologies, strengths, weaknesses, and finally propose a thorough detection method to tackle some of the identified challenges. As a result of the SQL Injection attacks, there is a need to develop a better SQL Injection detection system.

The main contributions to this work are:

- ✓ Authors identify the nature of SQL injection attacks, as well as develop a prevention mechanisms.
- ✓ Authors present a comprehensive framework for detecting all types of SQL attacks.

The remaining parts of this paper 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 examines them. Then, in section 4, methods employed in developing a compressive framework for SQL injection are presented, while section 5 presents the results and discussion. Finally, in section 6, the paper is concluded.

Existing methods for SQL injection detection and prevention

Several approaches have been proposed to detect and prevent SQL injection (SQLI) attacks; some focusing on statistical analysis [10-14] or dynamic analysis [15], [16], others used Hybrid approaches [17], [18]. These approaches are used for vulnerability analysis, scanning, mitigation, detection and prevention, and attack avoidance of web applications. There have been numerous studies to investigate vulnerability analysis through a detailed examination of a web application's security holes [19]. This has also been investigated in previous studies by vulnerability scanning tools used in [20]. Many authors have recognized detection and prevention as one of the best ways to avoid attack of web applications [21]. For instance, the following studies were conducted on the detection and prevention of SQL injection attacks (SQLIAs);

Authors in [22] examin all the SQL injection attack types and also different tools which can detect or prevent these attacks. All types of SQL injection attacks are discussed, as well as various techniques for detecting and preventing them. In [23], a comprehensive review of different types of SQL injection detection and prevention techniques was presented. Authors in [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. Authors in [25] examine the SQL injection attack types in an open-source database in MySQL. The detection and prevention of attacks were successfully performed using various approaches including machine learning, hybrid, and web-based tools. These approaches are most times adopted because database server security mechanisms such as authentication, authorization, server roles, database roles, system and object privileges are built-in and do not include all security features. Therefore, there is still a security issue when an attack with features other than its built-in function is performed.In addition, database types also differ from one another.

Machine Learning approach

Machine learning approaches can be used to develop vulnerability predictors, according to a variety of studies such as [26–29]. 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.

Hybrid (Statistic and Dynamic)

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

Developing a web-based framework

Many frameworks such as [33]–[35] have been developed and tested with various parameters. This has been presented by a number of authors in literature. Authors in [32] present a framework that can be used to handle tautology-based SQL injection attacks using the post-deployment monitoring technique. Authors in [34] introduce a novel traffic-based SQLIA detection and vulnerability analysis framework named DIAVA, which can proactively send warnings to tenants promptly. Authors in [36] propose a framework based on misuse and anomaly detection techniques to detect SQL injection attacks. Authors in [37] discuss a secure mechanism for protecting web applications from SQL Injection attacks by using framework and database firewall. Authors in [38] present a Runtime Monitoring Framework to detect and prevent SQL Injection Attacks on web applications. Authors in [39] present a cloud computing adoption framework (CCAF) security suitable for business clouds. Authors in [40] propose a 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. This helped raise the awareness of the seriousness of SQLIAs. On some given parameters, some of the methods performed well, while others did not. As a consequence, these frameworks detect the injected queries but they have no control over them. Looking at the reviewed literatures on SQL injection attacks, some gaps and shortcomings were identified. Implementing detection types for a single attack type has its own drawback in that it does not detect other attack types other than the one for which it was designed.

Common SQL injection attack

To address the issues raised in this paper, we provide a detailed overview of the various types of SQL injection attacks discovered to date. For each type of attack, we provide explanations and examples of how such attacks can be carried out, as well as explicit mitigation mechanisms. Finally, we propose a comprehensive framework that protects against all types of attacks.

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

Tautology attack

In this attack, the attacker attempts to use a conditional question argument to test the validity of a tautology attack. Using the WHERE clause, the attacker injects the condition and transforms it into a tautology that is always valid [41].

Table 1 describes the most common type of tautology attack, their nature, and the methods used to detect them.

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, there is need to study all forms of the tautology attack that leads to injection. Due to this, this study investigates the types of attacks and recommends approaches that are appropriate for each attack.

Union Query

In this category of attack, the UNION operator is only used for both queries which have the same form. The attacker constructs a SELECT statement that is similar to the original query [29]. To do so, in the first query, you must know the correct table name, as well as the number of columns and their data types.

As a result, two conditions must be met or a Union query attack will be launched, and each query returns the same number of columns [42]. If the data type of a column is incompatible with the string data, the injected query will fail. Table 2 presents details of the Union Injection Attack based on the nature of the attack.

Table 2: Union Injection Attack.

Type of injection	*Nature of attack*	*Recommended approach*
Union Query attack	Bypassing authentication, extracting data using union operation are characteristics of this attack	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.

Piggybacked Query

Data extraction, data addition or modification, denial of service, and remote command execution are all examples of attack determined by a Piggybacked Query. In this type of attack, an attacker attempts to inject additional queries into the original query. This form is distinct from others in that attackers attempt 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 this type of 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 prevent this form of attack, and run-time monitoring is not required.

Illegal/incorrect Query

This attack's goal includes identifying injectable parameters, performing database finger-printing, and extracting data. This attack helps an attacker to collect crucial information about the type and function of a Web application's back-end database [37]. The attack is thought to be a practice run for future attacks aimed at gathering information. This attack takes advantage of the fact that the default error pages [46] of application servers are frequently overly descriptive. As a result, Table 4 indicates the recommended approach for this attack.

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.

Stored Procedure Query

Users can save their features and access them at any time. Most collections of SQL queries include the ability to use them. The intruder executes the database's built-in stored procedures using malicious SQL Injection codes [47]. As a result, the cached stored procedure query plans are recompiled. The constraint of a Stored Procedure is that it can only be used in the database. Table 5 shows the best way to counter this attack.

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

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]. The attacker attempts to break into a site that has been sufficiently secured, so that when an injection is successful, there is no accessible feedback in the form of database error messages. Because database error messages do not provide feedback, the attacker must rely on another method to obtain a response from the database. Table 6 shows various types of attacks under inference queries.

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 help the attacker to make an informed decision about which injection method 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	The attack's main goal is to inject and execute system-level commands through a vulnerable program	Rule-based

.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 this attack from being detected	Machine Learning

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. Due to all the attacks discussed in this paper, an improved prevention and detection mechanism is required to prevent the Injection attacks.

Methodology

Proposed framework

To develop our framework, we studied the existing approaches and their attacking methods and limitations. To combat the challenges, we propose a comprehensive framework that encompasses solving all vulnerabilities that exist in previous works. In the proposed framework, the attacker must first open his browser to carry out the activity or perform an 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. The intruder must first get past the firewall checker.

Then web server accepts the user input by various mechanisms such as user input validation and then uses the input to create queries to the underlying database [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 the application server access again. However, in this situation, we suggested a model for evaluating whether or not the requested access involves a SQL injection which is shown in Figure 2.

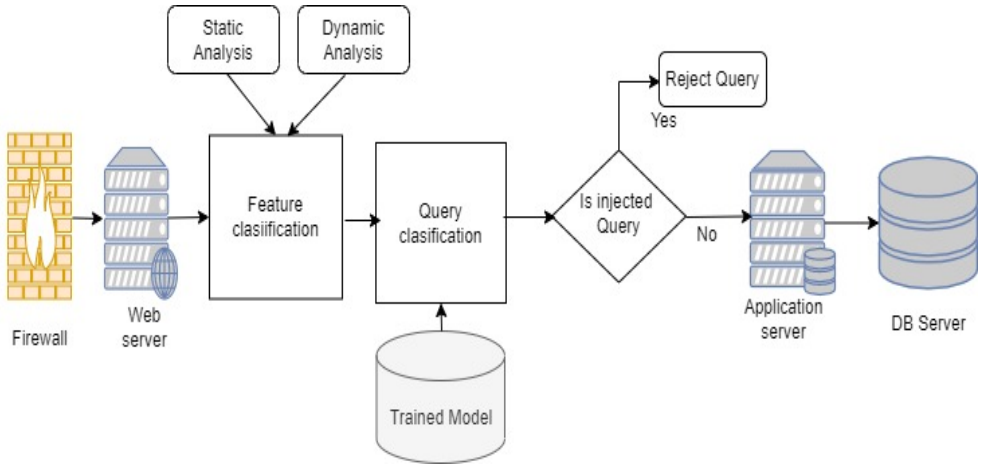


Fig. 2: Proposed frame work for SQL injection detection.

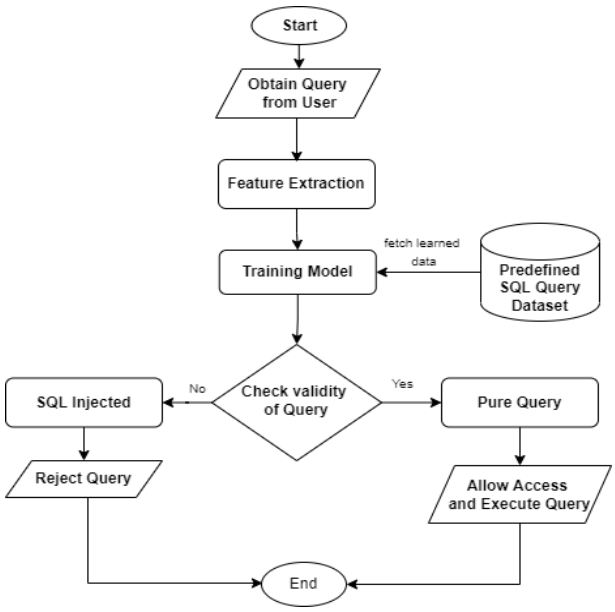
Before the classification of SQL queries, there were several stages. The first is feature extraction which is performed by using static and dynamic analysis to check whether the requested queries are injected with either approach. The classifier accepts queries and matches them with the trained dataset based on the requested query. Then the machine learning classifier accepts the extracted features and trains the model to identify the injected query. In literature SVM [50], [51], Decision tree, Naive Bayes [18], [52], [53], and other machine learning approaches [20], [54]–[57] have been proposed to solve the classification problem. The trained model comprises of stages such as pre-processing and feature extraction [58]. The classifiers are trained to recognize different types of SQL injection attacks according to the trained machine learning model and hybrid approach used in the feature extraction stage. 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 two methods (hybrid approach and machine learning approach) to obtain the best results possible when dealing with SQL query injection attacks.

The proposed method used supervised learning in order to train and test the model. Initially, the user acquires the dataset and then train and test are put as predefined datasets to train the model. This is followed by validation of the query. The flowchart of the proposed method is presented in Figure 3.

Results and discussion

Key insights and parameters for this study were obtained from the reviewed works which helped to analyze and evaluate the proposed methodology. Therefore, in this study, we employed three injection parameters which have various forms. The first is a user input field, which allows a web application 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 attacks parameters include all of the attack types mentioned in this paper. The acquired dataset was obtained from various sources, which include manual data collection via tutorials and public access (GitHub, hacker challenge websites), automatic tool payloads and recordings (SQLmap, OWASP Xenotix XSS Exploit Framework, XSSer, Metasploit Framework), and publicly available datasets (CIC IDS, NSL-KDD).



Fie 3: Flow chart of the training model.

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 information from real environment 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 is utilized as training data for the Naive Bayes, Decision Tree, SVM, and Hybrid methods to avoid SQL injection attacks. The dataset is trained before being utilized for classification. Python3 was used as the programming language and the experimentation was performed using a laptop. Keras framework based on TensorFlow was used with Python for the experimentation. The results of the classifiers during the training phase are presented in Table 8. The results show an accuracy rate of 99.16% for ANN and the hybrid approach has a rate of 99.6%, making it the best trained among the others. The training time for Naive Bayes and SVM, on the other hand, is extremely short as presented in Table 9.

Table 8: Results of different approaches on the training set

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

Table 9: Result of different approaches on the test set.

Approaches	Test set accuracy	Testing time (ms)
Naive Bayes	0.872	0.38
Decision tree	0.9413	0.67
SVM	0.9681	0.96
ANN	0.9852	3.11
Hybrid	0.9927	5.49

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

Conclusion

In this paper, we evaluated different techniques for detecting and preventing SQL Injection. To begin with, we defined the various forms of SQL injection that have been discovered thus far. After that, we evaluated the techniques under various consideration in terms of their ability to detect and prevent SQL attacks. We also investigated the various mechanisms and decided which techniques was capable of dealing with each mechanism. Thereafter, we identified each technique's specifications and developed a comprehensive framework to detect and prevent SQL injection attacks using a hybrid approach and machine learning techniques. Based on our models evaluation, we found that the hybrid approach and ANN are the best approaches to classify SQL injection. In this study, we used a small dataset for training and testing, but maximizing the dataset and implementing the model in practice is recommended for future studies.

***Authors**:*
Mr. Fitsum Gizachew Deriba, Department of Computer Science, Wachemo University Hossana, Ethiopia, E-mail: computer.fitsum@gmail.com;
Dr. Ayodeji Olalekan Salau, Department of Electrical/Electronics and Computer Engineering, Afe Babalola University Ado-Ekiti, Nigeria, E-mail: ayodejisalau98@gmail.com;
Mrs. Shaimaa Hadi Mohammed, Department of Computer Science, Summer University, Iraq;
Mr. Tsegay Mullu Kassa, Department of Information Technology, Wachemo University Hossana, Ethiopia, E-mail: tsegaymullu6@gmail.com;
Mr. Wubetu Barud Demilie, Department of Information Technology, Wachemo University Hossana, Ethiopia, E-mail: wubetubarud@gmail.com

REFERENCES

[1] M. A. Yunus *et al.*, “Review of SQL Injection : Problems and Prevention,” International Journal on Informatics Visualization, vol. 2, pp. 215–219, 2018. DOI:10.30630/JOIV.2.3-2.144
[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,” IEEE, 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,” *EURASIP J. on Info. Security*, 14, 2020. DOI: 10.1186/s13635-020-00113-y
[5] T. Qais, T. Mohammad, I. Jamil, *A novel method for preventing SQL injection using SHA-1 algorithm and syntax-awareness*, International conference on information and communication Technologies for Education and Training and international conference on Computing in Arabic (ICCA-TICET) (IEEE, Khartoum), pp. 1–4, 2017.

[6] A. Alazab, “New Strategy for Mitigating of SQL Injection Attack,” International Journal of Computer Applications, vol. 154, no. 11, pp. 1–10, 2016.
[7] A. Gurina and V. Eliseev, “Anomaly-Based Method for Detecting Multiple Classes of Network Attacks,” *Information*, vol. *10, no.* (3), 84; 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,” Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2020, 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.
[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,” International Journal of Engineering and Technology, 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: 1.3844/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] 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
[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 tchnology 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] Salau, A. O. and Jain, S. (2019). Feature Extraction: A Survey of the Types, Techniques, and Applications. *5th IEEE International Conference on Signal Processing and Communication (ICSC)*, Noida, India, pp. 158-164. DOI: 10.1109/ICSC45622.2019.8938371