Journal of Big Data

RESEARCH Open Access

Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques

Wubetu Barud Demilie[1*] and Fitsum Gizachew Deriba[2]

*Correspondence:
wubetubarud@gmail.com;
wubetuB@wcu.edu.et

[1] Department of Information Technology, Wachemo University, Hossana, Ethiopia
[2] Department of Computer Science, Wachemo University, Hossana, Ethiopia

Abstract

A web application is a software system that provides an interface to its users through a web browser on any operating system (OS). Despite their growing popularity, web application security threats have become more diverse, resulting in more severe damage. Malware attacks, particularly SQLI attacks, are common in poorly designed web applications. This vulnerability has been known for more than two decades and is still a source of concern. Accordingly, different techniques have been proposed to counter SQLI attacks. However, the majority of them either fail to cover the entire scope of the problem. The structured query language injection (SQLI) attack is among the most harmful online application attacks and often happens when the attacker(s) alter (modify), remove (delete), read, and copy data from database servers. All facets of security, including confidentiality, data integrity, and data availability, can be impacted by a successful SQLI attack. This paper investigates common SQLI attack forms, mechanisms, and a method of identifying, detecting, and preventing them based on the existence of the SQL query. Here, we have developed a comprehensive framework for detecting and preventing the effectiveness of techniques that address specific issues following the essence of the SQLI attacks by using traditional Navies Bayes (NB), Decision Trees (DT), Support Vectors Machine (SVM), Random Forests (RF), Logistic Regression (LR), and Neural Networks Based on Multilayer Perceptron (MLP), and hybrid approach are used for our study. The machine learning (ML) algorithms were implemented using the Keras library, while the classical methods were implemented using the Tensor Flow-Learn package. For this proposed research work, we gathered 54,306 pieces of data from weblogs, cookies, session usage, and from HTTP (S) request files to train and test our model. The performance evaluation results for training set in metrics such as the hybrid approach (ANN and SVM) perform better accuracies in precision (99.05% and 99.54%), recall (99.65% and 99.61%), f1-score (99.35% and 99.57%), and training set (99.20% and 99.60%) respectively than other ML approaches. However, their training time is too high (i.e., 19.62 and 26.16 s respectively) for NB and RF. Accordingly, the NB technique performs poorly in accuracy, precision, recall, f1-score, training set evaluation metrics, and best in training time. Additionally, the performance evaluation results for test set in metrics such as hybrid approach (ANN and SVM) perform better accuracies in precision (98.87% and 99.20%), recall (99.13% and 99.47%), f1-score (99.00% and 99.33%) and test set (98.70% and 99.40%) respectively than other ML approaches.

However, their test time is too high (i.e., 11.76 and 15.33 ms respectively). Accordingly, the NB technique performs poorly in accuracy, precision, recall, f1-score, test set evaluation metrics, and best in training time. Here, among the implemented ML techniques, SVM and ANN are weak learners. The achieved performance evaluation results indicated that the proposed SQLI attack detection and prevention mechanism has been improved over the previously implemented techniques in the theme. Finally, in this paper, we aimed to keep researchers up-to-date, with contributions, and recommendations to the understanding of the intersection between SQLI attacks and prevention in the artificial intelligence (AI) field.

Keywords: Deep learning, Detection, Hybrid, Machine learning, Prevention, SQLI attack, Web application

Introduction

Malware attacks, particularly SQLI attacks, are common in poorly designed web applications. This vulnerability has been known for more than two decades and is still a source of concern [1]. For many years, structured query language (SQL) has been the industry standard for dealing with relational database management systems (DBMS). Since the majority of applications for cyber-physical systems are safety–critical; misbehavior brought on by random errors or online attacks can severely limit their development [2, 3]. Therefore, it's crucial to safeguard cyber-physical systems from suffering this kind of attack.

SQLI attacks on data-driven web applications and systems, also known as SQLI attacks, have been a serious problem since it became common for internet web applications and SQL databases to be connected [4, 5, 6]. An SQLI attack occurs when an attacker takes advantage of a flaw in the web application's SQL implementation by submitting a malicious SQL statement through a fillable field. In other words, the attacker will insert code into a field to dump or alter data or gain access to the backend. As explained by [6] and [7], SQLI is a common attack vector that allows malicious SQL code to access hidden information by manipulating database backends and is regarded as one of the most dangerous injection attacks because it jeopardizes the main security services such as confidentiality, authentication, authorization, and integrity [8, 9]. This information could include sensitive business information, private customer information, or user lists. A successful SQLI attacker can lead to the deletion of entire databases, the unauthorized use of sensitive data, and the unintentional grant of administrative rights to a database.

The increased development and spread of web applications have also increased the number and severity of web attacks [10, 11]. According to [12], the most common vulnerability in web applications is injection. Injection attacks take advantage of a variety of flaws to deliver untrusted user input, which is then processed by a web application [13]. The SQLI attacks entail injecting (inserting) malicious SQL commands into input forms or queries to gain access to a database or manipulate its data (e.g. send the database contents to the attacker, modify or delete the database content, etc.) [14, 15]. Undeniably, most web applications today rely on a back-end database to store data collected from users and/or to retrieve information selected by users [16].

Forms and cookies are commonly used to interact with these users. Different hackers attempt to exploit this feature by injecting malicious code into the user inputs that will

later be used to construct the SQL queries. Improper validation of user inputs can result in the success of the SQLI attack, which can have disastrous consequences such as the deletion of the database or the collection of sensitive and confidential data from web application clients [17]. Several research works have addressed the SQLI attack due to its sensitive impact. Some of these works only attempt to detect SQLI after it has occurred and other works have attempted to prevent it from happening in the first place.

In this study, we looked at SQLI attacks that try to bypass the web application firewall and gain unauthorized access to confidential data. These attacks target the HTTP or HTTPS protocol. The victim system is normally not prepared to handle this input, which frequently leads to data leakage and/or the attacker receiving unauthorized access. In this instance, the attacker has access to and/or control over the data, which has an impact on all areas of security, including data availability, confidentiality, and integrity [2].

The SQL query that was maliciously injected is intended to extract or modify data from the database server. Successful injection can cause data loss and/or the total database to be destroyed, as well as authentication, bypass, and modifications to the database by inserting, changing, and/or deleting data. Additionally, such an assault could take control of the hosted OS and run commands on it, usually having greater negative effects [2, 18]. Therefore, organizations are seriously threatened by SQLI assaults.

Even though several techniques have been proposed to combat SQLI attacks, none of these solutions have addressed the full scope of the attacks. As a result, there were no solutions that can prevent or detect all types of SQLI attacks. Recently, researchers have attempted to with AI integrated techniques including deep learning (DL), machine learning (ML), and hybrid techniques to propose more sophisticated solutions [19].

Here, learning from past data reflecting an attack and/or regular data is typically used to build AI approaches to help with threat detection and prevention. Historical information can be used to interpret detected traffic, identify attack patterns, and even forecast future assaults before they happen [2, 20].

To grasp how SQL language can be abused, SQLI attackers and defenders need to understand how it functions [2]. The queries must be prepared in the SQL language and adhere to a specified syntax to retrieve data from databases or modify the data, such as:

*"SELECT * FROM authortable WHERE book_name = 'Advanced Database Systems'"*

The aforementioned search will provide all books with book_name "Advanced Database Systems." The queries are typically typed into a web browser and sent to the database management system [2].

What if the attacker in this case extends the original SQL query?

For example:

*"SELECT * FROM authortable WHERE book_name = 'Advanced Database Systems' OR '1'='1'"*

The above query will return all book names in the database, not just the book names labeled as "Advanced Database Systems," because the sentence '1 = 1' is always true. If the stored list of book names is not a secret and the previous example might not pose a problem [2]. If successful, it might return sensitive data, such as passwords, bank accounts, trade secrets, and personal information, which might be regarded as a privacy breach among other negative effects. However, it could be applied to value using different syntax.

According to certain studies, inserting code using 'OR' and a 'TRUE' assertion, such as '1 = 1', is referred to as "tautology" [21]. Other techniques besides tautology can be used, like when an attacker purposefully inserts an incorrect query to make the database server return a default error page. This default error page may contain important information that can help an attacker comprehend the database and craft a more sophisticated attack [2, 21]. In addition to numerous other techniques based on the same concept, incorrectly employing SQL syntax to extract or even edit the data in the targeted database, 'UNION' can also be used to extract information.

Given that SQLI operates in this manner, the question is how to recognize and stop this kind of attack by using DL, ML, and hybrid techniques. By instructing a classifier to develop the ability to recognize, detect, and subsequently prevent an attack, it can be used to support the detection and prevention of SQLI attacks. The classifier can be used to categorize new data, including traffic or data from log files, and is trained using various models. If the classifier is active, it will block data from reaching the database server; if it is passive, it will inform the administrator. Accordingly, three alternative learning techniques have been used to train the classifier to recognize and detect SQLI attacks [2, 22].

The first technique was unsupervised learning (UL), in which features were taken from data that had not been categorized, or data that had neither normal nor pathological labels. The classifier finds hidden structures in the unclassified dataset using information and the Bayesian probability theory. When data are unclassified, it is unclear if they are normal or abnormal (malicious). The UL can make use of a variety of methods, including clustering and density estimation [2, 22].

The classifier was trained using the second technique, supervised learning (SL), using a set of labeled training data. The output was known in advance because the input data were marked, i.e., normal or aberrant. To attain an acceptable classification accuracy, the procedure first comprises a straightforward mapping between the input training data and the known output, which was followed by continuous algorithm and weight change. The classifier was then tested using a test set of data; if the results were within an acceptable range of accuracy, the classifier was then ready to recognize novel data or data that had not been used in training or testing [2].

The fundamental problem with the SL was the time it took to create and label the training and test sets of data, especially for sophisticated attacks. The classification and regression techniques were used to classify the SL. The Bayesian networks, DT, SVM, K-nearest neighbors, and neural networks are some of the most used SL techniques. The third technique, known as semi-supervised learning, combines SL and UL techniques [2, 22]. Accordingly, by using SQLI, the attackers can alter the SQL statement by replacing the user's supplied data with their own data as depicted in Fig. 1.

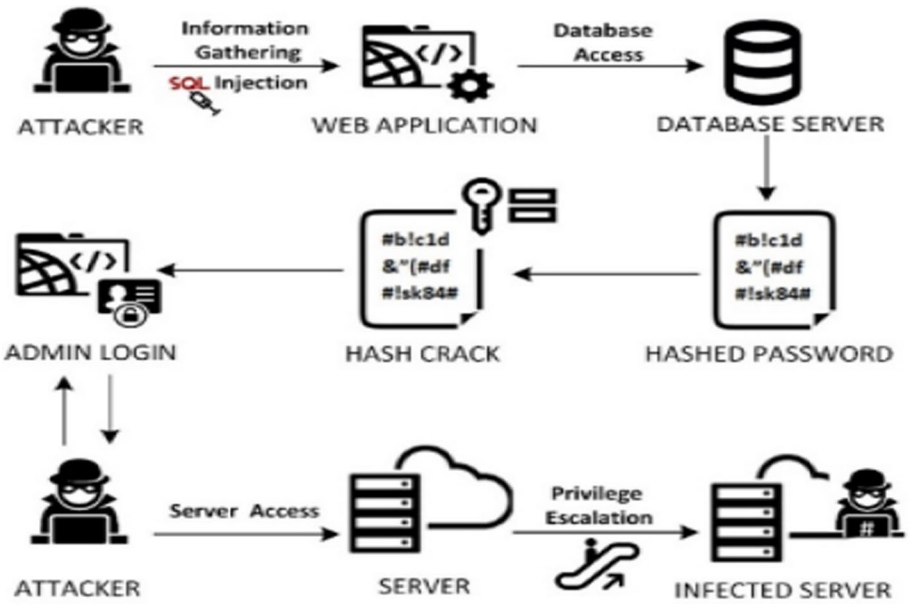

Fig. 1 The SQLI attacks [23]

For this proposed research work, we gathered 54,306 pieces of data from weblogs, cookies, session usage, and from HTTP (S) request files to train and test our model. We divided the acquired data into sections for our model's testing (30%) and training (70%). We used 38,014 of the total dataset for training and 16,292 for model testing. Among the datasets, 47,343 are genuine queries and 6,963 are malicious queries.

Accordingly, this work has the following major contributions.

- Review the recommended and state-of-the-art research works in DL, ML, and hybrid techniques for SQLI attacks that have been published in reputable databases.
- The different SQLI attack detection and prevention countermeasures are classified and discussed.
- Studying and identifying the nature of SQLI attacks and proposing prevention and detection mechanisms.
- Newly proposed solutions are described and discussed, such as those based on DL, ML, and hybrid techniques.
- Keep the researchers up-to-date, with contributions, and recommendations to the understanding of the intersection between SQLI attacks and prevention in the AI field.

Accordingly, the primary goal of this paper is to examine current SQLI attacks, identify their methodologies, strengths, and weaknesses, and finally propose a thorough detection and prevention method.

The rest of the paper is organized into different but interrelated sub-sections. The paper begins by discussing the related works in the "Related works" section, the SQLI query attacks overview in the "SQLI query attacks overview" section, existing methods for SQLI detection and prevention in the "Existing methods for SQLI detection and prevention" section, developing a web-based framework for SQLI attacks detection and

prevention in "Developing a web-based framework for SQLI attacks detection and prevention" section, most common attacks on SQLI in "Most common attacks on SQLI" section, proposed frameworks for SQLI detection and prevention in "Proposed frameworks for SQLI detection and prevention" section, result and discussion in "Result and discussion" section, and the conclusion and recommendation in "Conclusion and recommendation" section.

Related works

In this section, different published research works have been considered and included to indicate the research gaps in the area. The paper typically reviews and includes studies that have been published in reputable databases. Many researchers have demonstrated the use of DL, ML, and hybrid techniques to detect SQLI attacks [23].

A review of SQLI prevention in web applications has been presented in [1]. The authors have provided a summary of 14 different varieties of SQLI attacks and how they affect online applications. Their research's main objective was to investigate alternative SQLI prevention strategies and to offer an analysis of the most effective defense against SQLI attacks.

Authors in [2] have conducted a systematic literature review of 36 articles related to research on SQLI attacks and ML techniques. To classify different varieties of SQLI attacks, they have identified the most widely used ML techniques. Their finding revealed that few studies generated new SQLI attack datasets using ML tools and techniques. Similarly, their results showed that only a few studies focused only on using mutation operators to generate adversarial SQLI attack queries. In future work, the researchers aimed to cover the use of other ML and DL techniques to generate and detect SQLI attacks.

A comprehensive study on SQLI attacks, their mode, detection, and prevention has been presented in [4]. The authors have identified how attackers of this kind might exploit such a weakness and execute weak code as well as a strategy to mitigate such detrimental effects on database systems. The researchers' investigation revealed that web operations were frequently used for online administrations ranging from high levels of informal communication to managing transaction accounts and dealing with sensitive user data. The real issue, however, was that this data was exposed to attacks because of unauthorized access, where the attackers gained entry to the system using various hacking and cracking techniques with very malicious motives. The attacker can use more sophisticated queries and creative tactics to get around authentication while also gaining total control over both the server and the web application. Many cutting-edge algorithms have been developed up to this point to encrypt data queries to defend against such attacks by structuring desirable query modification plans. In the paper, they worked together to discuss the history of injection attacks, different forms of injection attacks, various case studies, and defenses against SQLI attacks, along with an appropriate illustration.

In the work of [5], a survey on SQLI attack detection and prevention has been presented. The research, according to the authors, might help laypeople comprehend SQL and its hazards. It also helps researchers and programmers who wanted to learn about all the problems that still plague web applications and what strategies can be employed

to stop SQLI attacks. From the researcher's perspective, it was anticipated that if web application developers adhered to the strategies provided in their study, the online applications would be safe from such damaging attacks.

Detecting web attacks with end-to-end DL was presented in [10]. Three new insights into the study of autonomous intrusion detection systems have come from this work. Firstly, they assessed whether a method based on the resilient software modeling tool (RSMT), which autonomously monitors and describes the runtime behavior of web applications, was feasible for detecting web attacks. A low-dimensional representation of the raw features with unlabeled request data was used to recognize anomalies by computing the reconstruction error of the request data, and they have also described how RSMT trains a stacked denoising autoencoder to encode and reconstruct the call graph for end-to-end DL.

Secondly, they have described how RSMT trains a stacked denoising autoencoder to encode and reconstruct the call graph for end-to-end DL, where a low-dimensional representation of the raw features with unlabeled request data is used to recognize anomalies by computing the reconstruction error of the request data. Thirdly, they have examined the outcomes of empirically testing RSMT on artificial datasets as well as real-world applications that have been intentionally made vulnerable. Finally, the findings demonstrated that the suggested method could efficiently and accurately identify attacks, such as SQLI, cross-site scripting, and deserialization, with a minimum of labeled training data and domain knowledge.

According to [11], SQLI is a common and challenging network attack that can cause inestimable loop-breaking and loss to the database, and how to detect SQLI statements was one of the current research hotspots. Here, how to detect SQLI statements was one of the current research hotspots. As described by the authors, SQLI is a frequent and difficult network assault that can result in immeasurable loop-breaking and loss to the database. An SQLI detection model and technique based on deep neural networks were developed based on the data properties of SQL statements. The main technique used in this case was word pausing the data to turn it into word vectors, then forming a sparse matrix and feeding it into the model for training. Next, a multi-hidden layer deep neural network model with the ReLU function was built, the traditional loss function was optimized, and a dropout method was added to increase the generalizability of this model and over 96% of the final model's accuracy was achieved. Finally, the proposed technique successfully addressed the issues of overfitting in ML and the requirement for manual screening to extract features, which significantly increases the accuracy of SQLI detection by comparing the experimental results with conventional ML and LSTM algorithms.

Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications has been presented in [8]. To identify XQuery injection and parameter tampering vulnerabilities in online applications powered by native extensible markup language (XML) databases, a black-box fuzzing approach has been proposed. A working prototype of XiParam was created and put to the test on weak web applications that used BaseX, a native XML database, as their backend. The experimental analysis amply proved that the prototype was successful in preventing both XQuery injection and parameter tampering vulnerabilities from being detected.

In the work of [16], an SQLI attack detection and prevention technique using DL has been presented. Based on extensive local and international research, the authors have suggested an SQLI detection method that uses NLP and DL frameworks and does not rely on a background rule base. By allowing the machine to automatically pick up on the language model characteristics of SQLI attacks, the strategy has increased accuracy, decreased false alarm rates, and provided some protection against attacks that were never discovered in advance.

Detection of SQLI attacks has been presented, tested, and compared to 23 ML classifiers using MATLAB [23]. They generated their own datasets, into which they injected abnormal SQL syntax. They checked and manually verified the SQL statements. A total of 616 SQL statements were used to train the test classifiers. They have used ML techniques such as "coarse KNN, bagged trees, linear SVM, fine KNN, medium KNN, RUS boosted trees, subspace discriminant, boosted trees, weighted KNN, cubic KNN, linear discriminant, medium tree, subspace KNN, simple tree, quadratic discriminant, cubic SVM, fine Gaussian SVM, cosine KNN, complex tree, logistic regression, coarse Gaussian SVM, medium Gaussian, and SVM". The five best models in terms of accuracy were determined to be ensemble boosted, bagged trees, linear discriminant, cubic SVM, and fine Gaussian SVM. They have tested their proposed technique and the results showed that their technique was able to detect the SQLI attack with an accuracy of 93.8%.

The authors of [24] have proposed a model called ATTAR to detect SQLI attacks by analyzing web access logs to extract SQLI attack features. The features were chosen based on access behavior mining and a grammar pattern recognizer. The main target of this model was the detection of unknown SQLI statements that had not been previously used in the training data. Five ML techniques were used for training: NB, random forest, SVM, ID3, and k-means. The experimental results showed that the accuracy of the models based on random forest and ID3 achieved the best results in detecting SQLI attacks.

The authors of [25] have proposed a hybrid CNN-BiLSTM-based model for SQLI attack detection. The authors presented a detailed comparative analysis of different types of ML techniques used for the detection of SQLI attacks. The CNN-BiLSTM approach provided an accuracy of approximately 98%, compared with other described ML techniques.

The authors of [26] have presented an ML classifier to detect SQLI vulnerabilities in PHP code. Multiple ML techniques were trained and evaluated, including random forest, logistic regression, SVM, multilayer perceptron (MLP), LSTM, and CNN. The authors have found that CNN provided the best precision of 95.4%, while a model based on MLP achieved the highest recall 63.7%, and the highest f-measure of 74.6%.

The authors of [27] have proposed an adaptive deep forest model (ADF) with the integration of the AdaBoost technique. AdaBoost stands for adaptive boosting, which is a statistical classification technique, and the deep forest model is a layered model based on a deep neural network. The adaptive deep forest model proposed in [25] achieved high efficiency, comparable to that of traditional ML models, such as decision trees, and better performance compared with regular deep neural network models, such as RNN and CNN.

The authors of [28] have created a dataset using symbolic finite automata to train a classifier to detect SQLI attacks. The generated data were labeled, and training was

conducted with an SL model with ML techniques of two-class support vector machine (TC SVM) and two-class logistic regression (TC LR). The generated models were evaluated using a receiver operating characteristic (ROC) curve.

The authors of [29] have proposed an SQLI detection method using ensemble learning techniques and NLP to generate a bag-of-words model used to train a random forest classifier. The prediction was also considered in this research to improve the detection ability of the classifier. In this study, DT, NB, SVM, and KNN classification models were also trained to classify the same testing dataset, and their performances were compared with that of the proposed method. The experimental results showed that the proposed method achieved better accuracy, higher TPR, and lower FNR than the other four classifiers. The evaluation metrics were used to measure the performance of the classifier. The measurements were based on a confusion matrix, accuracy, precision, true-positive rate, false-positive rate, true-negative rate, false-negative rate, receiver operating characteristic curve, and area under the curve.

The authors of [30] have developed a dataset by gathering and combining a large number of smaller datasets. The generated dataset was labeled, and the learning model was SL. They trained seven ML models: DT, AdaBoost, random forest, optimized linear, TensorFlow linear, deep ANN, and a boosted trees classifier. Then, they compared the seven techniques in terms of performance and accuracy. The results showed that the random forest classifier outperformed all other classifiers and achieved an accuracy of 99.8%. The paper also compared the performance of different ML models in detecting SQLI attacks.

The authors of [31] have proposed a novel approach to the detection of SQLI attacks using human agent knowledge transfer (HAT) and TD ML techniques. In this model, an ML agent acted as a maze game to differentiate between normal SQL queries and malicious SQL queries. If the incoming SQL query was an SQLI attack query, then it gained more rewards and was deemed an SQLI attack query before achieving the final state. Finally, the ML technique has achieved an accuracy of 95%.

The authors of [32] have proposed a detection system based on two techniques. The first detection method was based on pattern matching, which is the same as a signature-based detection system whereby the classifier has a database of SQL attack signatures and only inspects the HTTP URL in an attempt to find a match. The second detection method used was based on ML techniques. To build this model, the authors have collected malicious data and trained the classifier with these data by extracting the features representing attacks. They have used techniques such as SVM, NB, and K-nearest neighbor. The performance of the classifier was measured using the total cost ratio (TCR).

The authors of [33] have trained an SVM to detect malicious SQL queries by modeling the WHERE clause of a query as an interaction network of tokens and computing the centrality of the nodes. Node centralities were used to quantify the degree of importance or centrality of a node in the network. The experimental results obtained on a dataset collected from five web applications using some automated attack tools confirmed that three of the centrality measures used in this study can effectively detect SQLI attacks with minimal impact on performance.

The authors of [15] have proposed an LSTM-based SQLI attack detection method, which can automatically learn the effective representation of data and has a strong advantage to confront complex high-dimensional massive data. Additionally, from the

standpoint of penetration, this paper has provided an injection sample development method based on data transmission channels. This technique can produce legitimate positive samples and explicitly simulate SQLI attacks. The strategy, in the researcher's opinion, can successfully address the over-fitting issue brought on by a lack of sufficient positive samples. The experimental findings revealed that the suggested method outperformed numerous similar classical ML techniques and widely used DL techniques in terms of improving the accuracy of the SQLI attack detection and reducing the false positive rate. Finally, the experimental results showed that the accuracy, precision, and f1-score of the proposed method were all above 92%.

The authors of [34] have proposed a framework for SQLI prevention via server-side scripting using ML and compiler platforms. A dataset of 1100 samples of SQL commands were trained in four ML techniques such as boosted decision tree, DT, SVM, and an artificial neural network. The results indicated that the DT technique has achieved the highest prediction efficiency among the tested models.

The author of [35] has used the AdaBoost technique to detect SQLI attacks. In this study, the data were converted into stumps, which were classified as weak stumps providing less weight to the output, or strong stumps providing the highest weight in the overall output. The experimental result showed that the proposed technique accurately and effectively detected injection attacks.

The authors of [36] have proposed a method for classifying dynamic SQL queries as either attacks or normal based on a web profile prepared during the training phase. NB, SVM, and parse tree techniques were used for the classification process. The overall detection rate using the two datasets was 91% and 90%, respectively.

The author of [37] has designed a method to detect malicious SQL queries. The DT technique was used for the classification processes to detect different levels of SQLI. The proposed model maintained an accuracy of more than 98% in detecting SQLI attacks and an accuracy of 92% in classifying the level of attack as simple, unified, or lateral.

The authors of [38] have presented a simple method for SQLI attack detection based on an artificial neural network. First, a large amount of SQLI data were analyzed to extract the relevant features. Then, a variety of neural network models, such as MLP and LSTM, were trained. The experimental results showed that the detection rate of MLP was better than that of LSTM.

The authors of [39] have automatized the process of exploiting SQLI attacks through reinforcement learning agents. In this study, the problem was modeled as a Markov decision process. The experimental results showed that reinforcement learning agents can be used in the future to perform security assessment and penetration testing.

The authors of [40] have presented a detection method by modeling SQL queries as a graph of tokens and utilized the centrality measure of tokens to train single and multiple SVM classifiers. The system was tested using directed and undirected graphs with different SVM classifiers. The experimental results demonstrated that the proposed technique can effectively identify malicious SQL queries.

The authors of [41] have presented a model of a two-class support vector machine (TCSVM) to predict binary labeled outcomes concerning whether an SQLI attack was positive or negative in a web request. This model has intercepted web requests at the proxy level and applied ML predictive analytics to predict SQLI attacks.

The authors of [42] have presented a novel approach for classifying SQL queries. A gap-weighted string subsequence kernel technique was used to compute the similarity metric between the query strings. Then, the SVM was trained on the similarity metrics to determine whether the query strings were normal or malicious. The proposed approach was evaluated using many datasets and achieved an accuracy of 92.48%.

The authors of [43] have presented a new approach to the construction of a dataset with a NoSQL query database. Six classification techniques were trained and evaluated to identify SQLI attacks, which included: DT, SVM, random forest, KNN, neural network, and multilayer perceptron. The experimental results showed that the last two techniques obtained an accuracy of 97.6%.

The authors of [44] have trained a progressive neural network model with an NB classifier to successfully detect SQLI attacks. Progressive neural networks were trained using parameters such as error-based, time-based, SQL query, and union-based SQLI attacks. The proposed method has achieved an accuracy of 97.897%.

The authors of [45] have proposed a hybrid approach using tree-vector kernels in SVM to learn SQL statements. The authors used both the parse tree structure of SQL queries and the query value similarity characteristic to distinguish between malicious and benign queries. The results confirmed the benefit of incorporation to efficiently and accurately identify abnormal queries.

The work [14] has presented the detection of SQLI behaviors using word vectors and long short-term memory (LSTM). A unique technique for detecting SQLI attacks based on a word vector of SQL tokens and LSTM neural networks was presented in this paper. In the suggested approach, SQL query strings were first syntactically broken down into tokens, after which a word vector of SQL tokens was built using the likelihood ratio test, and finally, an LSTM model was trained using sequences of token word vectors. They created a tool called WOVSQLI to implement the suggested method, and it was tested using a set of data from several sources. The performed experiments showed that WOVSQLI was capable of reliably detecting SQLI attacks. Finally, the results of the experiment showed that the proposed tool achieved an accuracy of 98.60%.

The authors of [46] have proposed a DL-based approach to detect SQLI attacks in network traffic. The proposed approach selected only the target features needed by the model to be trained using a deep belief network (DBN) model. The authors also employed test data to test the performance of different models, including LSTM, CNN, and MLP. According to the experimental results, DBN achieved an accuracy of 96%.

The authors of [47] have proposed a framework that combined the EDADT (efficient data adaptive decision tree) technique and the SVM classification technique to detect SQLI attacks. The used dataset was created using the MovieLens dataset system for movie recommendations, which included user login and movie details. The experimental results showed that the proposed approach achieved an accuracy of 99.87%.

The authors of [48] have proposed a method for detecting SQLI using the NB ML technique. The authors have applied a tokenization process to break the query into meaningful elements called tokens. Then, the list of tokens became an input for further classification processes. The result of the NB technique was analyzed using precision, recall, and accuracy.

Evading web application firewalls through adversarial machine learning has been presented in [49]. They have presented WAF-A-MoLE, a tool that models the presence of an adversary. This tool leverages a set of mutation operators that alter the syntax of a payload without affecting the original semantics. The researchers have evaluated the performance of the tool against existing WAFs, which they have trained using their publicly available SQL query dataset. Finally, they showed that WAF-A-MoLE bypasses all the considered ML-based WAFs.

Deep semantic learning for testing SQLI has been presented in [50]. The paper has proposed a deep natural language processing-based tool, dubbed DeepSQLI, to generate test cases for detecting SQLI vulnerabilities. Through adopting DL based neural language model and sequence of words prediction, DeepSQLI was equipped with the ability to learn the semantic knowledge embedded in SQLI attacks, allowing it to translate user inputs (or a test case) into a new test case, which was semantically related and potentially more sophisticated. The experiments were conducted to compare DeepSQLI with SQLmap, a state-of-the-art SQLI testing automation tool, on six real-world web applications that were of different scales, characteristics, and domains. The empirical results demonstrated the effectiveness and the remarkable superiority of DeepSQLI over SQLmap, such that more SQLI vulnerabilities can be identified by using a less number of test cases, whilst running much faster.

Behind an application firewall, are we safe from SQLI attacks has been presented in [51]. The paper was focused on web application firewalls and SQLI attacks. They have presented an ML-based testing approach to detect holes in firewalls that let SQLIattacks bypass. In the beginning, the approach can automatically generate diverse attack payloads, which can be seeded into inputs of web-based applications, and then submit them to a system that was protected by a firewall. Incrementally learning from the tests that were blocked or passed by the firewall, their approach can then select tests that exhibit characteristics associated with bypassing the firewall and mutate them to efficiently generate new bypassing attacks. In the race against cyber attacks, time was vital. Being able to learn and anticipate more attacks that can circumvent a firewall promptly was very important to quickly fix or fine-tune the firewall. They have developed a tool that implements the approach and evaluated it on ModSecurity, a widely used application firewall. The results they obtained suggest good performance and efficiency in detecting holes in the firewall that could let SQLI attacks go undetected.

Automatic detection of NoSQLI using SL has been presented in [52]. They have developed a tool for detecting NoSQLI using SL. To the best of their knowledge, their developed training dataset on NoSQLI was the first of its kind. They manually designed important features and apply various SL techniques. Their tool has achieved a 93.00 f1-score as established by tenfold cross-validation. They also applied their tools to a NoSQLI generating tool, NoSQLMap, and find that their tool outperforms Sqreen, the only available NoSQLI detection tool, by 36.25% in terms of detection rate. The proposed technique was also shown to be database-agnostic achieving similar performance with injection on MongoDB and CouchDB databases.

A framework for SQLI investigations detection, investigation, and forensics has been presented in [53]. This paper has proposed a framework of SQLI investigation architecture and has proved its feasibility in fighting against SQLI attacks. An effective and

efficient approach was also proposed to prosecute SQLI aggressors and keep them away from abusing the database.

The development of a compressive framework using ML Approaches for SQLI attacks has been presented in [54]. The paper investigates the most common SQLI attack forms, their mechanisms, and a method of identifying them based on the existence of the SQL query. Furthermore, they have proposed a comprehensive framework for determining the effectiveness of the proposed techniques in addressing a variety of issues based on the type of attack using DL, ML, and hybrid techniques. A thorough examination of the model using a test set revealed that the hybrid approach and ANN outperform NB, SVM, and DT in terms of classifying injected queries. The NB, on the other hand, outperforms the other approaches in terms of web loading time during testing. The results showed an accuracy of 99.16% for ANN and the hybrid technique has an accuracy of 99.6%, making it the best trained among the others. As a result, the proposed method improved the detection and prevention of SQLI attacks. They used a small dataset for training and testing in this study, but maximizing the dataset and implementing the model in practice was recommended for future researchers.

The development of a compressive framework using ML Approaches for SQLI attacks has been presented in [54]. The paper investigates the most common SQLI attack forms, their mechanisms, and a method of identifying them based on the existence of the SQL query. Furthermore, they have proposed a comprehensive framework for determining the effectiveness of the proposed techniques in addressing a variety of issues based on the type of attack using DL, ML, and hybrid techniques. A thorough examination of the model using a test set revealed that the hybrid approach and ANN outperform NB, SVM, and DT in terms of classifying injected queries. The NB, on the other hand, outperforms the other approaches in terms of web loading time during testing. The results showed an accuracy of 99.16% for ANN and the hybrid technique has an accuracy of 99.6%, making it the best trained among the others. As a result, the proposed method improved the detection and prevention of SQLI attacks. They used a small dataset for training and testing in this study, but maximizing the dataset and implementing the model in practice was recommended for future researchers.

According to the recommendation of [54], the detection and prevention rate of the system can be improved by increasing the training and testing dataset and by using the recommended techniques. So, we have proposed this research work to increase the systems detection and prevention rate of SQLI attacks in different web applications.

SQLI query attacks overview

An SQL is a language developed to manage data stored in relational databases [12]. It allows users to access, modify, and delete data. Many web applications and websites keep all of their data in SQL databases. SQL commands can also be used to run OS commands in some cases.

Web attacks are one of the major topics to be investigated in this study. Even though there are numerous web attacks, SQLI is one of the most common and will be among the top five web attacks in 2021, according to the OWASP report [55, 56]. This attack grants attackers complete access to databases containing sensitive information [1, 10, 57, 58]. As in a common understanding, a web application has three levels [54]: The first

presentation layer collects user feedback and shows the user the processing results. The presentation layer directly communicates with the user. The second control layer, the server script, processes data entered by the user and sends the results to the database layer. The database layer sends the processed data to the control layer, which then sends it to the presentation layer for the user to view [59–62]. As a result, data processing occurs on the control layer in the web application, which can be implemented in a variety of server scripting languages. The database (DB) layer finally saves and retrieves the data. The database stores and manages all sensitive web application data. Because this layer is directly connected to the control layer and has no security checks, if the control layer is successfully attacked, data in the database can be exposed and modified. The general concept of web-based architecture is depicted in Fig. 2.

The difficulty in perceiving the injected query at the database layer necessitates a system that controls and filters the query at the presentation layer based on predetermined parameters [63]. Various studies have been conducted to identify and prevent the injected queries. The bulk of them, however, do not identify all types of SQLI, but they fared better on a handful in the statistical or dynamic portions. Vulnerabilities in web applications can exist if the sanitization function does not correctly sanitize user input. The static analysis cannot tell whether or not the input has been sanitized properly. The vulnerabilities often go unnoticed due to such flaws in static assessments.

The SQL prevent checks produced queries for those parameters and provides an alarm when a hypertext transfer protocol (s) (HTTP (S)) request parameter influences the syntax structure of a query [64]. Different techniques are used to track user inputs, and a profile of caring queries has been created. This technique has a high rate of false positives and false negatives due to the inability of the application that generates the query to encapsulate input [65]. As a result, there is still a gap in the SQLI attack forms. Here, we have proposed this work to fill such gaps by using the recommended techniques and state-of-the-art research works accordingly.

Existing techniques for SQLI detection and prevention

Several techniques for detecting and preventing SQLI have been proposed, with some focusing on statistical analysis [4, 66, 67, 68, 69, 70] or dynamic analysis[71, 72], others on Hybrid approach [32, 73]. These techniques are used for web application vulnerability

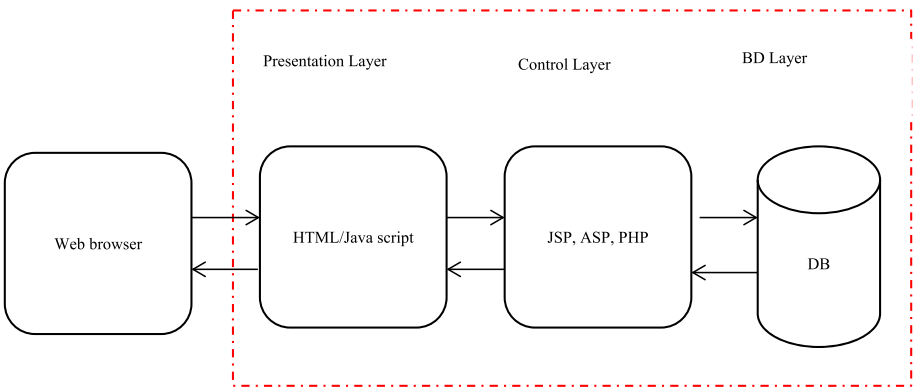

Fig. 2 Web application architecture [54]

analysis, scanning, detection and prevention, mitigation, and attack avoidance. Numerous studies have been conducted to investigate vulnerability analysis by conducting a thorough examination of a web application's security flaws [74]. It has also been investigated in previous studies by vulnerability scanning tools used in [75]. Many authors have recognized detection and prevention are the best way to avoid attacks from web applications [76]. The detection and prevention of attacks were successfully established by various techniques including DL, ML, and hybrid. The techniques have been described in the following sections.

DL techniques

SQLI is a common and difficult attack on web applications, systems, and network security issues. Deep or convolutional neural networks (CNN), can be used in a wide variety of threat detection and prevention scenarios [10, 11]. Code injection is the most common and damaging attack, ranking first on the OWASP vulnerabilities list [16, 17]. Accordingly, the detection and prevention of code injection attacks, which were previously done using signature or pattern-based recognition techniques, has recently been supplemented by the use of advanced ML techniques.

ML techniques

Several types of research implied that ML techniques [12, 77–79] 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 to new injections. A vulnerability analysis method needs to be able to adapt when more advanced security threats are discovered. The ML technique allows for re-training to respond to new vulnerability trends [16].

Hybrid techniques

The hybrid injection detection and prevention system (HIDPS) uses both ML classifiers and other statistical techniques to prevent and detect the rescues of SQLI attacks from different web applications and systems [32]. Accordingly, some of the previous research has used hybrid techniques [64, 73, 80]. This can be done by comparing the structure of the queries to detect attacks. Initially, it detected if a dynamically generated query 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 were followed. A new approach is therefore needed for SQLI attacks [18, 81].

Developing a web-based framework for SQLI attacks detection and prevention

Frameworks have become an essential part of web development because, as web application standards rise, so does the complexity of the technology required [82]. It's completely unreasonable to reinvent the wheel with such sophisticated techniques. As a result, using frameworks endorsed by thousands of developers worldwide is a very sound approach to developing rich and interactive web applications. Because a web application has a backend (server-side) and a frontend (client-side) for both the backend and

frontend frameworks. Many frameworks such as [53, 83, 84] have been developed and tested with various parameters.

Authors of [85] proposed a framework based on misuse and anomaly detection techniques to detect SQLI attacks. The research of [86] discusses a secure mechanism for protecting web applications from SQLI attacks by using a framework and database firewall. An author of [81] presents a framework that can be used to handle tautology-based SQLI attacks using the post-deployment monitoring technique. The authors of [87] evaluate runtime monitoring frameworks to detect and prevent SQLI attacks on web applications. The authors of [88] present a cloud computing adoption framework (CCAF) security suitable for business clouds. The authors of [89] propose SQLI intrusion detection framework as a service for SaaS providers, structured query language injection identity as a service (SQLI IDaaS), which allows a SaaS provider to detect structured query language injection attacks (SQLIAs) targeting several SaaS applications without reading, analyzing, or modifying the source code. To raise the tenants' awareness of the seriousness of SQLIAs. The research work of [83] introduces a novel traffic-based SQLIA detection and vulnerability analysis framework named (DIAVA), which can proactively send warnings to tenants promptly. Some of them perform well on the given parameters, while others do not. As a result, while these frameworks detect injected queries, they have no control over them. However, a closer examination of the literature on the aforementioned SQLI attack reveals numerous gaps and shortcomings.

Most common attacks on SQLI

As we know, SQL is a programming language that is used to create, update, and access data in a database. A hacker can intentionally cause the application to fail, delete data, steal data, or gain unauthorized access by carefully crafting SQL commands [90]. To address the aforementioned issue, we provide a detailed overview of the various types of SQLI 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 is resistant to all types of attacks for detection and prevention.

Tautology attack

The attacker attempts to use a conditional query argument to test always true in the tautology attack, such as $(1=1)$ or $(--)$. The attacker injects the condition and transforms it into a tautology that is always valid using the WHERE clause [91, 92]. This type of attack is commonly used to access databases without requiring authentication on websites [1].

For example:

```
SELECT * FROM books WHERE book_id= '1' OR '1=1' --' AND book_published_date=
'25/10/2022';
```

The most common type of tautology attack, the nature of the attack, and the approach used to detect them are described below in Table 1.

Table 1 Common tautology attacks [54]

Type of injection	Nature of attack	Approach for detection
String SQLI	Bypassing authentication, identifying injectable parameters using string data type, extracting data	Rule-based
Numeric SQLI	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

Table 2 Union injection attack [54]

Type of injection	Nature of attack	Recommended techniques
Union query attack	Bypassing authentication, extracting data using union operation	Rule-based

Accordingly, the SQL query results turn the original condition into a tautology, allowing an unauthorized user, for example, to access all records in the database table. Guardium detects and prevents many variants of tautological statements in database requests. Previous studies were limited to investigating the most common tautology attack, but it is still necessary to investigate all types of tautology attacks that lead to injectables. As a result, the study investigates the different types of attacks and recommends a strategy for each one.

Union query

In this category of attack, the UNION operator is only used if both queries have the same form, the attacker constructs a SELECT statement that is similar to the original query [1, [79]. To do so, it must be known that the correct table name, as well as the number of columns and their data types from the first query. As a result, two conditions may be satisfied, or an attack on the union query will be launched, and each query returns the same number of columns [92, 93]. If the data type of a column is incompatible with string data, the injected query will fail.

For example:

```
SELECT * FROM booktable WHERE book_id= '5667' UNION SELECT * FROM
authortable WHERE author_id= 'main' AND password= '7668';
```

Based on the nature of the attack the suggested approach is described below in Table 2. Mostly the second query in a union is malicious [94], 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

The piggybacked query attack concatenates more query statements onto the initial query ";" [1]. This technique is especially risky since it enables an attacker to insert virtually any SQL command. Data extraction, addition or modification of data, denial of service (DoS), and remote command execution are all examples of determined attacks [16]. In this type of attack, an attacker attempts to inject additional queries into the original query. Unlike other forms, attackers attempt to add new and distinct queries that "piggyback" on the original query rather than changing it [92, 95, 96].

For example:

```
SELECT book_name FROM booktable WHERE book_id= 'book1' AND book_name = 0;
DROP booktable;
```

As a result, several SQL queries are sent to the database. Table 3 states the nature and appropriate approach used for the attack. These types of criminal behavior can be avoided by first locating the correct SQL query through appropriate validation or by employing various detection mechanisms. This type of attack can be avoided using static analysis and no run-time monitoring is required.

Illegal or incorrect query

This kind of attacker takes advantage of a database query that was improperly executed [1]. It will show database error messages, which frequently provide crucial facts that enable an attacker to learn the application's database specifics. The attack goal includes identifying injectable parameters, performing database fingerprinting, and extracting data. This attack assists an attacker in gathering critical information concerning the nature and function of the back-end database of a web application [86, 92]. 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 on application servers are frequently excessively descriptive [97].

For example:

```
SELECT * FROM booktable WHERE book_id= '4546' AND book_published_year =
'2022' AND CONVERT (char, no);
```

Due to that the recommended approach is shown in Table 4.

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

Table 3 Piggybacked query [54]

Type of injection	Nature of attack	Recommended technique
Piggybacked query	Adding or altering data, performing DoS, and executing remote commands are all examples of data extraction	ML

Table 4 Illegal or incorrect query [54]

Type of injection	Nature of attack	Recommended technique
Illegal/incorrect Query	Error messages ignored by the client are used to locate useful data, allowing the backend database to be injected more easily	ML

Table 5 Stored procedure query [54]

Type of injection	Nature of attack	Recommended techniques
Stored procedure Query	Performing privilege escalation, denial of service, and remote command execution	Rule-based

Stored procedure query

Here, an attacker can use this technique to modify the database's stored procedures [1]. Both authorized and unauthorized users will receive true or false results from the process. The users can save their features and use them whenever they want. A collection of SQL queries are provided with the feature to use it. The intruder uses malicious SQL codes to execute the database's built-in stored procedures [92, 98]. 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.

For example:

```
SELECT author_name FROM authortable WHERE author_name= 'author1' AND author_id= ' '; SHUTDOWN;
```

The best method to overcome this attack is described below in Table 5.

Inference query

In this attack, the query is recast as an operation and executed based on the answer to a true or false question about database data values [92, 99, 100]. For this method of injection, attackers attempt to break into a site that has been sufficiently protected that when an injection is successful and there is no accessible feedback in the form of database error messages. Since database error messages are not available to provide feedback and attackers must rely on another approach to get a response from the database.

For example:Consider (malicious parameter (inference attack on SQL server. Here, 1; if SYSTEM_USER='sa' SELECT 1/0 ELSE SELECT 5) [101].

QUERY GENERATED (two possible outcomes for the injected IF).

```
SELECT name, email FROM members WHERE id=1; IF SYSTEM_USER='sa' SELECT 1/0 ELSE SELECT 5;
```

Different forms of attack under inference query are shown in Table 6.

Table 6 Common inference query attack [54]

Types of attack	Nature of attack	Recommended technique
Blind SQLI [91]	Collect valuable data by inferring from the page's answers after asking the server a set of true/false questions	ML
Timing Attack[96]	Observe the response time, which will assist the attacker in making an informed decision about which injection approach to use	ML
Database Backdoor Attack	Set a trigger to collect the user's feedback and send it to his or her e-mail address	ML
Command SQLI	Injecting and executing system-level commands via a vulnerable program is the essence of the attack	Rule-based

Table 7 Alternative encoding query [54]

Type of Injection	Nature of Attack	Recommended Technique
Alternative Encoding Query	Safe protective coding and automatic prevention systems are used to keep it from being detected	ML

Alternative encoding query

In this attack, the injected text is changed to avoid detection by protective coding practices as well as several automated prevention techniques. This attack is used in conjunction with others [92, 102]. To intend their attack they use the regular expression [76]. Authors in [92] have explained this type of SQLI attack with examples. This implies that they do not provide a unique way to target an application; rather, they are an enabling technology that allows attackers to circumvent detection and prevention strategies and exploit the vulnerabilities depicted in Table 7.

Proposed frameworks for SQLI detection and prevention

Because of the nature of the attack and the need for detection and prevention mechanisms, a more systematic and theoretical analysis of SQLI attacks is required. To develop our framework, we have investigated existing techniques, as well as their attacking methods and flaws accordingly. As a result, we propose a comprehensive framework that addresses all vulnerabilities identified in the previous research works. To carry out the activity, the attacker must first open his browser and if the application is open, the intruder either enters his password into the application or requests authorization to access the web service via the internet. The intruder must first get past the firewall checker to proceed. The web server then accepts user input through various mechanisms, such as user input validation, and uses the input to generate queries to an underlying database [64, 93]. This can be accomplished by identifying injection parameters, determining the type and version of a web application's database, and determining the database schema. If the attacker was granted permission based on the request, he will request application server access again. However, in this case, we proposed a model

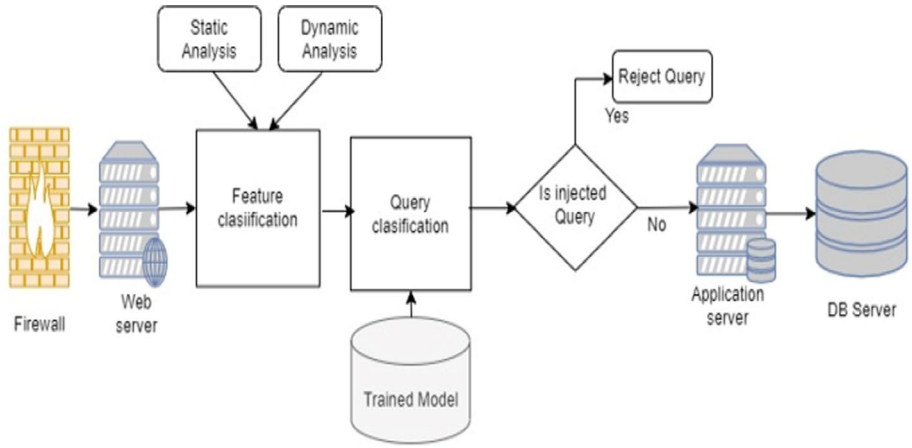

Fig. 3 Proposed framework for SQLI attack detection and prevention [54]

for the detection and prevention of determining whether or not the requested access involves SQLI, as shown in Fig. 3.

There were several stages preceding the classification of SQL queries. The first feature extraction is done by comparing the static and dynamic analysis to see if the requested queries are injected with either approach. Based on the query, the classifier accepts it and matches it with the trained dataset. The extracted feature is then accepted by the ML classifier, which trains the model to identify the injected query. The SVM [103, 104], DT, NB [73, 105, 106], and other algorithms in ML techniques [75, 107–111] are used to solve classification algorithms. The trained model passes all stages such as preprocessing and feature extraction.

As a result, during the feature extraction steps, the classifiers will be trained to recognize various types of SQLI attacks based on the given trained ML model and hybrid approach. Based on the trained pre-fetched and trained dataset, the model matches the pattern of each line query requested. If the SQL query contains one or more qualified attacks, the model will either reject the request or send it to the application and database servers to perform the requested operation if the query is pure SQL with no injection. As a result, we propose developing a new architecture based on ML and hybrid approaches to achieve the best possible results when dealing with SQLI query attacks.

Result and discussion

In this study, we used three injection parameters in various forms. The first is through a user input field, which allows a web application to use HTTP (S) POST and GET to request information from a backend database, and the second is through cookies, which can be used to restore a client's state information when they return to a web application. An attacker can exploit this vulnerability to change cookies and submit them to the database server if a web application uses the contents of cookies to construct SQL queries. Finally, a server variable can be created by analyzing session usage information and recognizing browsing behaviors. Because attackers can forge the values in HTTP (S) and network headers by entering malicious input into the

Table 8 Performance evaluation for the training set

No	Techniques	Evaluation Metrics				
		Precision	Recall	F1-score	Training set accuracy	Training time (in sec.)
	NB	88.33%	87.89%	88.11%	89.40%	08.73
	DT	93.09%	92.75%	92.92%	95.70%	53.01
	SVM	97.15%	98.02%	97.58%	98.80%	19.06
	RF	97.28%	96.00%	96.64%	95.30%	09.48
	ANN	99.05%	99.65%	99.35%	99.20%	19.62
	Hybrid	99.54%	99.61%	99.57%	99.60%	26.15

Table 9 Performance evaluation for test set

No	Techniques	Evaluation Metrics				
		Precision	Recall	F1-score	Test set accuracy	Testing time (in msec.)
	NB	87.53%	86.37%	86.95%	87.20%	01.73
	DT	91.72%	90.84%	91.28%	94.80%	06.87
	SVM	96.40%	95.61%	96.00%	97.30%	03.8
	RF	94.35%	93.06%	93.70%	93.40%	05.19
	ANN	98.87%	99.13%	99.00%	98.70%	11.76
	Hybrid	99.20%	99.47%	99.33%	99.40%	15.33

application's client-end or by crafting their request to the server, logging these variables to a database without sanitization could result in SQLI vulnerability. Accordingly, all the attacks sent to the server are logged and saved as attack log data in the database. Furthermore, attack log data is divided into two categories: attacks and normal data.

Using various ML techniques, we trained and assessed vulnerability classifier models to determine which approach performed the best. The set of algorithms includes traditional NB, DT, SVM, RF, LR, and Neural Networks Based on MLP and hybrid techniques that are used for our study. The ML algorithms were implemented using the Keras library, while the classical methods were implemented using the Tensor Flow-Learn package.

We evaluated the performance of the models using ten-fold cross-validations, where the dataset was divided into ten different partitions and the final accuracy result was recorded.

During the training and testing of the selected techniques, we can get multiple classifiers, and we need to evaluate the performance of each classifier using appropriate evaluation metrics, from which the best one is selected. The samples can be combined according to the real target category and the category predicted by the classification model to obtain the following four cases:

True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Let TP, TN, FN, and FP denote their corresponding sample numbers, then obviously TP + TN + FN + FP = the total number of samples.

The confusion matrix of the classification results by taking each class as positive samples separately is shown in Tables 8 and 9.

For classification models, the evaluation criteria are precision, recall, f1-score, and training/test set accuracy as described in Eqs. (1–4). Since positive and negative sample imbalance is very common in the field of SQLI attack detection and prevention, it is unreasonable to use only accuracy rate as the evaluation metric, so the evaluation metric used is f1-score as the detection and prevention classifier performance in addition to detection accuracy (accuracy), check-all rate (recall), and check-accuracy rate (precision). the f1-score is used as a comprehensive evaluation criterion for classifier performance [11].

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP + TN}{TP + FN} \tag{3}$$

$$F1 - score = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{4}$$

The results of techniques in the training and testing phases are described in Tables 8 and 9 respectively.

According to the study shown in Table 8 and Fig. 4, the hybrid approach (ANN and SVM) performs better accuracies in precision (99.05% and 99.54%), recall (99.65% and

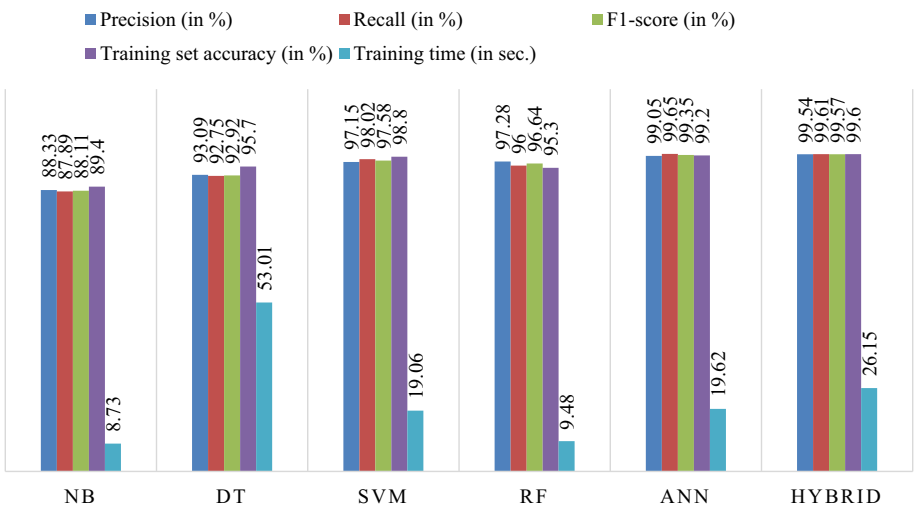

Fig. 4 Performance evaluation for the training set

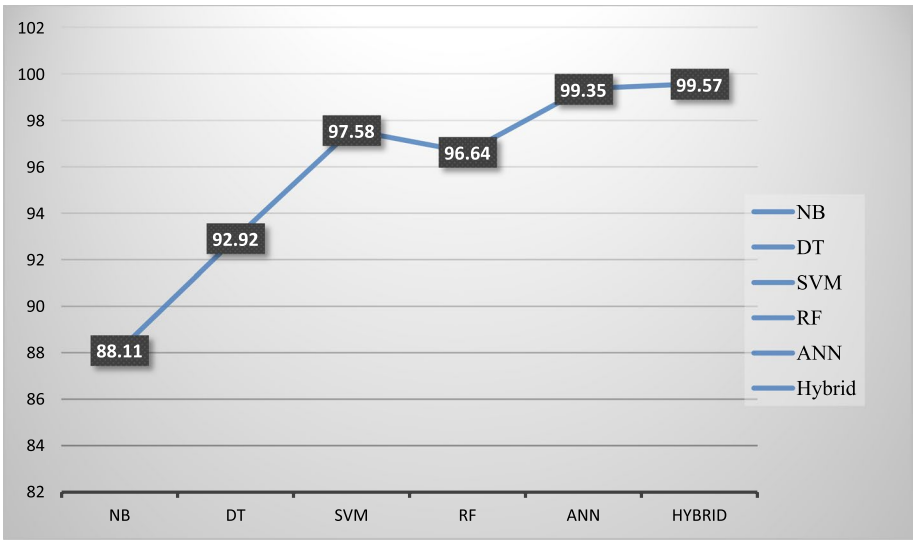

Fig. 5 Performance distribution of the techniques in the training set (f1-score)

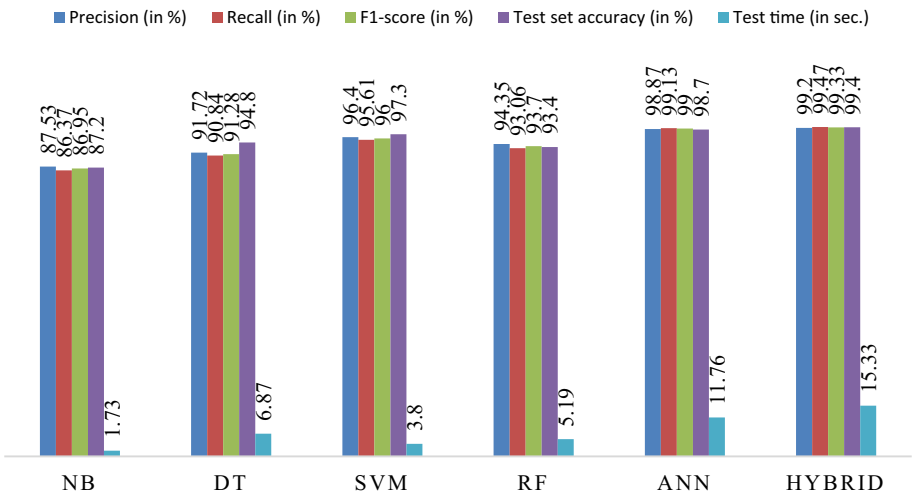

Fig. 6 Performance evaluation for test set

99.61%), f1-score (99.35% and 99.57%), and training set (99.20% and 99.60%) respectively than other ML approaches. However, their training time is too high (i.e., 19.62 and 26.16 s respectively) for NB and RF. Accordingly, the NB technique performs poorly in accuracy, precision, recall, f1-score, training set evaluation metrics, and best in training time.

From all the implemented techniques for SQLI attack detection and prevention, we have achieved the best performance in hybrid techniques as depicted in Table 8 and Fig. 4 for the given training sets.

Additionally, as indicated in Fig. 5, it can be observed that the distribution of the f1-score value is 88.11%, 92.92%, 97.58%, 96.64%, 99.35%, and 99.57% for the NB, DT, SVM, RF, ANN, and for hybrid techniques respectively.

According to the study shown in Table 9 and Fig. 6, the hybrid approach (ANN and SVM) performs better accuracies in precision (98.87% and 99.20%), recall (99.13% and 99.47%), f1-score (99.00% and 99.33%) and test set (98.70% and 99.40%) respectively than other ML approaches. However their test time is too high (i.e., 11.76 and 15.33 ms respectively). Accordingly, the NB technique performs poorly in accuracy, precision, recall, f1-score, test set evaluation metrics, and best in training time. Here, among the implemented ML techniques SVM and ANN are weak learners.

From all the implemented techniques for SQLI attack detection and prevention, we have achieved the best performance in hybrid techniques as depicted in Table 9 and Fig. 6 for the given test sets.

Additionally, as indicated in Fig. 7, it can be observed that the distribution of the f1-score value is 86.95%, 91.28%, 96.00%, 93.70%, 99.00%, and 99.35% for the NB, DT, SVM, RF, ANN, and for hybrid techniques respectively.

Even if there has been no research work on detecting and preventing SQLI attacks that have produced 100% accuracy, it is recommended future researchers in the area can use hybrid techniques with great consideration of large-scale datasets to increase the performance in the evaluation metrics (detection and prevention rates) of the systems. Accordingly, the performance of any SQLI attack systems detection and prevention rate depends on a detailed examination of the datasets, including the size of the data, the platform from which the data has been collected, and the selected techniques. Finally, as to the recommendation of [54], the detection and prevention rate of the system has been improved by increasing the training and testing datasets and by using the recommended techniques accordingly.

Conclusion and recommendation

SQLI is the most dangerous web application attacker. This type of attacker poses a significant risk to web applications and this may have major implications for privacy and security issues. Web application attacks are becoming increasingly common and severe. A

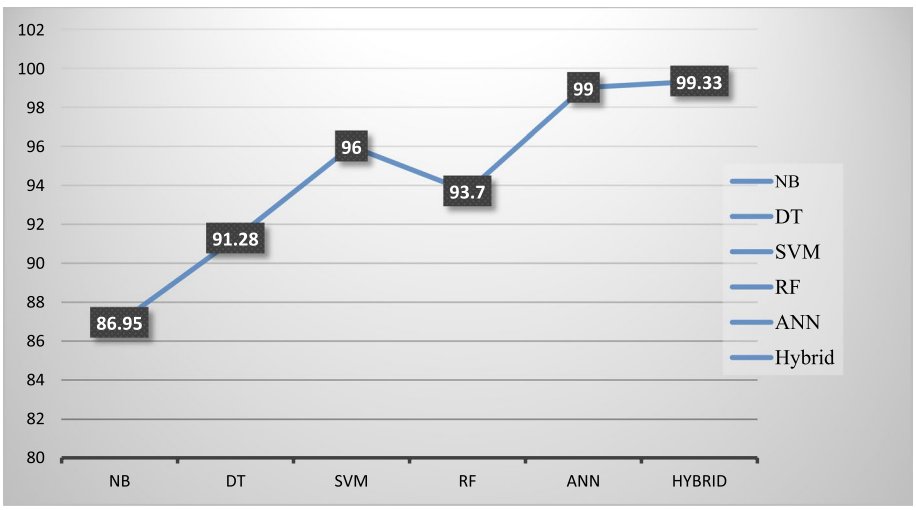

Fig. 7 Performance distribution of the techniques in the test set (f1-score)

large amount of data available on the internet motivates hackers to launch novel attacks. Several studies have been conducted to mitigate this attack, either by preventing it at an early stage or by detecting it when it occurs. We evaluated various strategies for detecting and preventing SQLI. Firstly, we have defined the different types of SQLI attacks that have been discovered thus far.

The techniques under consideration were then evaluated in terms of their ability to detect and prevent SQLI attacks. We identified the most commonly used DL, ML, and hybrid techniques to detect and prevent all types of SQLI attacks. We also looked into the various mechanisms and determined which techniques could deal with the detection and prevention of such SQLI attacks from different web applications. Then, using ML and hybrid techniques, we identify the specifications for each technique and develop a comprehensive framework for detecting and preventing SQLI attacks. We investigated that hybrid and ANN are the best techniques for classifying SQLI based on our model performance evaluation. The performance evaluation results for training set in metrics such as the hybrid approach (ANN and SVM) perform better accuracies in precision (99.05% and 99.54%), recall (99.65% and 99.61%), f1-score (99.35% and 99.57%), and training set (99.20% and 99.60%) respectively than other ML approaches. However, their training time is too high (i.e., 19.62 and 26.16 s respectively) for NB and RF. Accordingly, the NB technique performs poorly in accuracy, precision, recall, f1-score, training set evaluation metrics, and best in training time. Additionally, the performance evaluation results for test set in metrics such as hybrid approach (ANN and SVM) perform better accuracies in precision (98.87% and 99.20%), recall (99.13% and 99.47%), f1-score (99.00% and 99.33%) and test set (98.70% and 99.40%) respectively than other ML approaches. However, their test time is too high (i.e., 11.76 and 15.33 ms respectively). Accordingly, the NB technique performs poorly in accuracy, precision, recall, f1-score, test set evaluation metrics, and best in training time. Here, among the implemented ML techniques SVM and ANN are weak learners. Finally, in this research work, we aimed to keep researchers up-to-date, with contributions, and recommendations to the understanding of the intersection between SQLI attacks and prevention in the AI field. Here, maximizing the dataset and running with different techniques in a real-world environment is recommended for future researchers.

Acknowledgements
Not applicable.

Author contributions
WBD: Prepared the manuscript including analysis, data curation, visualization, conceptualization, methodology, and writing of the original draft. FGD: Performs tasks including conceptualization, implementation, validation, and review. Both authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that no competing interest.

Received: 2 February 2022 Accepted: 25 December 2022
Published online: 30 December 2022

References

1. Johny JHB, Nordin WAFB, Lahapi NMB, Leau YB. SQL Injection prevention in web application: a review. In: Communications in computer and information science, vol. 1487 CCIS, no. January. 2021. p. 568–585. https://doi.org/10.1007/978-981-16-8059-5_35.
2. Alghawazi M, Alghazzawi D, Alarifi S. Detection of sql injection attack using machine learning techniques: a systematic literature review. J Cybersecur Privacy. 2022;2(4):764–77.
3. Han S, Xie M, Chen HH, Ling Y. Intrusion detection in cyber-physical systems: techniques and challenges. IEEE Syst J. 2014;8(4):1052–62.
4. Dasmohapatra S, Priyadarshini SBB. A comprehensive study on SQL injection attacks, their mode, detection and prevention. 2021. p. 617–632. https://doi.org/10.1007/978-981-16-3346-1_50.
5. Hu J, Zhao W, Cui Y. A survey on SQL injection attacks, detection, and prevention. In: ACM international conference on proceeding series, no June. 2020. p. 483–488. https://doi.org/10.1145/3383972.3384028.
6. Blog. What is SQL injection attack? Definition & FAQs|Avi networks.
7. Imperva. SQL (structured query language) injection. Imperva. 2021.
8. Deepa G, Thilagam PS, Khan FA, Praseed A, Pais AR, Palsetia N. Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications. Int J Inf Secur. 2018;17(1):105–20. https://doi.org/10.1007/s10207-016-0359-4.
9. Dizdar A. SQL injection attack: real life attacks and code examples. 2022.
10. Pan Y, et al. Detecting web attacks with end-to-end deep learning. J Internet Serv Appl. 2019. https://doi.org/10.1186/s13174-019-0115-x.
11. Zhang W, et al. Deep neural network-based SQL injection detection method. Secur Commun Networks. 2022;2022:1–9. https://doi.org/10.1155/2022/4836289.
12. Pattewar T, Patil H, Patil H, Patil N, Taneja M, Wadile T. Detection of SQL injection using machine learning: a survey. Int Res J Eng Technol (IRJET). 2019;6(11):239–46.
13. Banach Z. Most dangerous food pathogens. 2022.
14. Fang Y, Peng J, Liu L, Huang C. WOVSQLI: detection of SQL injection behaviors using word vector and LSTM. In: ACM international conference on proceeding series. 2018. p. 170–174. https://doi.org/10.1145/3199478.3199503.
15. Li Q, Wang F, Wang J, Li W. LSTM-based SQL injection detection method for intelligent transportation system. IEEE Trans Veh Technol. 2019;68(5):4182–91. https://doi.org/10.1109/TVT.2019.2893675.
16. Chen D, Yan Q, Wu C, Zhao J. SQL injection attack detection and prevention techniques using deep learning. J Phys Conf Ser. 2021;1757(1):012055. https://doi.org/10.1088/1742-6596/1757/1/012055.
17. Abaimov S, Bianchi G. A survey on the application of deep learning for code injection detection. Array. 2021;11(June):100077. https://doi.org/10.1016/j.array.2021.100077.
18. Son S, McKinley KS, Shmatikov V. Diglossia: detecting code injection attacks with precision and efficiency. Proc ACM Conf Comput Commun Secur. 2013;2:1181–91. https://doi.org/10.1145/2508859.2516696.
19. Yan R, Xiao X, Hu G, Peng S, Jiang Y. New deep learning method to detect code injection attacks on hybrid applications. J Syst Softw. 2018;137:67–77. https://doi.org/10.1016/j.jss.2017.11.001.
20. P. Vähäkainu and M. Lehto, "Artificial intelligence in the cyber security environment," *Proc. 14th Int. Conf. Cyber Warf. Secur. ICCWS2019 Artif.*, 2019.
21. Singh G, Kant D, Gangwar U, Singh AP. SQL injection detection and correction using machine. |In: Emerging ICT bridging future—proceedings of the 49th annual convntion of Computer Society of India, vol. 1. 2015. p. 435–442. https://doi.org/10.1007/978-3-319-13728-5.
22. Marashdeh Z, Suwais K, Alia M. A survey on SQL injection attack: detection and challenges. 2021.
23. Hasan M, Balbahaith Z, Tarique M. Detection of SQL injection attacks : a machine learning approach. In: 2019 international conference on electrical computing technologies and applications. 2019.
24. Gao H, Zhu J, Liu L, Xu J, Wu Y, Liu A. Detecting SQL injection attacks using grammar pattern recognition and access behavior mining. In: 2019 IEEE international conference on energy internet. 2019. p. 493–498. https://doi.org/10.1109/ICEI.2019.00093.
25. Gandhi N, Patel J, Sisodiya R, Doshi N, Mishra S. A CNN-BiLSTM based approach for detection of SQL injection attacks. In: 2021 international conference on computational intelligence and knowledge economy. 2021. p. 378–383.
26. Zhang K. A machine learning based approach to identify SQL injection vulnerabilities. In: 2019 34th IEEE/ACM international conference on software engineering and automation. 2019. p. 1286–1288. https://doi.org/10.1109/ASE.2019.00164.
27. Li Q, Li W, Wang J, Cheng M. A SQL injection detection method based on adaptive deep forest. IEEE Access. 2019;7:145385–94.
28. Uwagbole SO, Buchanan WJ, Fan L. An applied pattern-driven corpus to predictive analytics in mitigating SQL injection attack. In: 2017 seventh international conference on emerging security technologies. 2017. https://doi.org/10.1109/EST.2017.8090392.
29. Ahmed M, Uddin MN. Cyber attack detection method based on nlp and ensemble learning approach. In: 2020 23rd international conference on computer information technology (ICCIT). 2020. https://doi.org/10.1109/ICCIT51783.2020.9392682.
30. Tripathy D, Gohil R, Halabi T. Detecting SQL injection attacks in cloud saas using machine learning. 2020.
31. Kulkarni CC, Kulkarni SA. Human-agent knowledge transfer applied to web security. 2013. https://doi.org/10.1109/ICCCNT.2013.6726770.

32. Makiou A, Begriche Y, Serhrouchni A. Hybrid approach to detect SQLi attacks and evasion techniques. In: collaborative 2014—proceedings of the 10th IEEE international conference on collaborative computing, networking, applications and worksharing. 2015. p. 452–456. https://doi.org/10.4108/icst.collaboratecom.2014.257568.
33. Kar D, Sahoo AK, Agarwal K, Panigrahi S, Das M. Learning to detect SQLIA using node centrality with feature selection. In: 2016 International conference on computer analysis security trends. 2017. https://doi.org/10.1109/CAST.2016.7914933.
34. Kamtuo K, Soomlek C. Machine learning for SQL injection prevention on server-side scripting. 2016.
35. Sivasangari A, Jyotsna J, Pravalika K. SQL injection attack detection using machine learning algorithm. 2021. https://doi.org/10.1109/icoei51242.2021.9452914.
36. Das D, Sharma U, Bhattacharyya DK. Defeating SQL injection attack in authentication security: an experimental study. Int J Inf Secur. 2019;18:1–22. https://doi.org/10.1007/s10207-017-0393-x.
37. Kasim Ö. An ensemble classification-based approach to detect attack level of SQL injections. J Inf Secur Appl. 2021. https://doi.org/10.1016/j.jisa.2021.102852.
38. Tang P, Qiu W, Huang Z, Lian H, Liu G. Detection of SQL injection based on artificial neural network. Knowl-Based Syst. 2020. https://doi.org/10.1016/j.knosys.2020.105528.
39. Erdődi L, Sommervoll ÅÅ, Zennaro FM. Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents. J Inf Secur Appl. 2021. https://doi.org/10.1016/j.jisa.2021.102903.
40. Kar D, Panigrahi S, Sundararajan S. SQLiGoT: detecting SQL injection attacks using the graph of tokens and SVM. 2016. p. 206–225. https://doi.org/10.1016/j.cose.2016.04.005.
41. Uwagbole SO, Buchanan WJ, Fan L. Applied machine learning predictive analytics to SQL injection attack detection and prevention. 2017. https://doi.org/10.23919/INM.2017.7987433.
42. McWhirter PR, Kifayat K, Shi Q, Askwith B. SQL Injection Attack classification through the feature extraction of SQL query strings using a Gap-Weighted String Subsequence Kernel. J Inf Secur Appl. 2018;40:199–216. https://doi.org/10.1016/j.jisa.2018.04.001.
43. Mejia-Cabrera HI, Paico-Chileno D, Valdera-Contreras JH, Tuesta-Monteza VA, Forero MG. Automatic detection of injection attacks by machine learning in NoSQL databases. 2021. p. 23–32. https://doi.org/10.1007/978-3-030-77004-4_3.
44. Pathak RK, Mohit, Yadav V. Handling SQL injection attack using progressive neural network. 2020. https://doi.org/10.1007/978-981-15-9671-1_20.
45. Wang Y, Li Z. SQL injection detection via program tracing. IDCS 2012, LNCS 7646. 2012. p. 264–265
46. Zhang H, Zhao B, Yuan H, Zhao J, Yan X, Li F. SQL injection detection based on deep belief network. 2019. p. 1–6.
47. Priyaa BD, Devi MI. Hybrid SQL injection detection system. 2016. https://doi.org/10.1109/ICACCS.2016.7586332.
48. Joshi A, Geetha V. SQL Injection detection using machine learning. 2014. https://doi.org/10.1109/ICCICCT.2014.6993127.
49. Demetrio L, Valenza A, Costa G, Lagorio G. WAF-A-MoLE: evading web application firewalls through adversarial machine learning. 2020. p. 1745–1752. https://doi.org/10.1145/3341105.3373962.
50. Liu M, Li K, Chen T. DeepSQLi: deep semantic learning for testing SQL injection. 2020. p. 286–297. https://doi.org/10.1145/3395363.3397375.
51. Appelt D, Nguyen CD, Briand L. Behind an application firewall, are we safe from SQL injection attacks? 2015. https://doi.org/10.1109/ICST.2015.7102581.
52. Islam MRU, Islam MS, Ahmed Z, Iqbal A, Shahriyar R. Automatic detection of NoSQL injection using supervised learning. 2019. https://doi.org/10.1109/COMPSAC.2019.00113.
53. Kao DY, Lai CJ, Su CW. A framework for SQL injection investigations: detection, investigation, and forensics.In: Proceedings of the 2018 IEEE international conference on system, man, and cybernetics SMC. 2018. p. 2838–2843. https://doi.org/10.1109/SMC.2018.00483.
54. Deriba FG, Salau AO, Mohammed SH, Kassa TM, Demilie WB. Development of a compressive framework using machine learning approaches for SQL injection attacks. PRZEGLĄD ELEKTROTECHNICZNY. 2022;1(7):181–7. https://doi.org/10.15199/48.2022.07.30.
55. OWASP. OWASP top 10_2021. 2021.
56. Kingthorin. SQL injection _ OWASP Foundation. 2022.
57. Amin M, et al. Review of SQL injection : problems and prevention. JOIV Int J Inform Vis. 2018;2:215–9.
58. Kumar A, Binu S. Proposed method for SQL injection detection and its prevention. Int J Eng Technol. 2018;7:213–6.
59. Hendita G, Kusuma A. Analysis of SQL injection attacks on website service. bit-Tech. 2018;1(1):26–33.
60. Abikoye OC, Abubakar A, Dokoro AH, Akande ON. A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm. EURASIP J Inf Secur. 2020. https://doi.org/10.1186/s13635-020-00113-y.
61. Yun X, Wen W. Cyber security. 2018.
62. Alazab A. New strategy for mitigating of SQL injection attack. Int J Comput Appl. 2016;154(11):1–10.
63. Gurina A, Eliseev V. Anomaly-based method for detecting multiple classes of network attacks. Information. 2019. https://doi.org/10.3390/info10030084.
64. Jahanshahi R, Doupé A, Egele M. You shall not pass : mitigating SQL injection attacks on legacy web applications. 2020. p. 445–457
65. Medeiros I, Beatriz M, Neves N, Correia M. SEPTIC: detecting injection attacks and vulnerabilities inside the DBMS. IEEE Trans Reliab. 2019;68(3):1168–88. https://doi.org/10.1109/tr.2019.2900007.
66. Gupta MK, Govil MC, Singh G. 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;2014:9–13. https://doi.org/10.1109/ICRAIE.2014.6909173.
67. Fu X, Lu X, Peltsverger B, Chen S, Qian K, Tao L. A static analysis framework for detecting SQL injection vulnerabilities. In: Proceedings of the international conference on computer software application, vol. 1, no. Compsac. 2007. p. 87–94. https://doi.org/10.1109/COMPSAC.2007.43.

68. Alenezi M, Javed Y. Open source web application security: a static analysis approach. In: Proceedings of the 2016 international conference on engineering and MIS, ICEMIS 2016. 2016. https://doi.org/10.1109/ICEMIS.2016.7745369.
69. Spoto F, et al. Static identification of injection attacks in Java. ACM Trans Program Lang Syst. 2019;41(3):1–58.
70. Basutakara BS, Jeyanthi PN. A review of static code analysis methods for detecting security flaws. J Univ Shanghai Sci Technol. 2021;23(06):647–53. https://doi.org/10.51201/jusst/21/05320.
71. Das D, Sharma U, Bhattacharyya D. An approach to detection of SQL injection attack based on dynamic query matching. Int J Comput. 2010;1(25):28–34.
72. Nanda S, Lam LC, Chiueh TC. Dynamic multi-process information flow tracking for web application security. In: Proceedings of the 8th ACM/IFIP/USENIX international conference on middleware 2007, Middleware'07. 2008. p. 1–20. https://doi.org/10.1145/1377943.1377956.
73. Hernawan FY, Hidayatulloh I, Adam IF. Hybrid method integrating SQL-IF and Naïve Bayes for SQL injection attack avoidance. J Eng Appl Technol. 2020;1(2):85–96.
74. Senthamil Preethi K, Murugan A. Analysis of vulnerability detection tool for web services. Int J Eng Technol. 2018;7:773–8.
75. Techniques P, et al. Design and implementation of SQL injection vulnerability scanning tool. J Phys Conf Ser. 2020. https://doi.org/10.1088/1742-6596/1575/1/012094.
76. Kumar BJS, Anaswara PP. Vulnerability detection and prevention of SQL injection. Int J Eng Technol. 2018;7:16–8.
77. Zolanvari M, Member S, Teixeira MA, Member S, Gupta L, Member S. Machine learning based network vulnerability analysis of industrial internet of things. 1–14.
78. Azman MA, Marhusin MF, Sulaiman R, Sains U, Marhusin MF, Sains U. Machine learning-based technique to detect SQL injection attack. J Comput Sci. 2021. https://doi.org/10.3844/jcssp.2021.296.303.
79. Krishnan SSA, Sabu AN, Sajan PP, Sreedeep AL. SQL injection detection using machine learning, vol 11, no 3. p. 300–310.
80. Kumar BJS, Pujitha K. Web application vulnerability detection using hybrid string matching algorithm. Int J Eng Technol. 2018;7:106–9.
81. Dharam R, Shiva SG. Runtime monitors for tautology based SQL injection attacks. In: Proceedings of the 2012 international conference on cyber security cyber warfare digital forensic, cybersecurity. 2012. p. 253–258. https://doi.org/10.1109/CyberSec.2012.6246104.
82. Goel A. Best web development tools in 2022. Ramotion. 2022.
83. Gu H, et al. DIAVA: a traffic-based framework for detection of SQL injection attacks and vulnerability analysis of leaked data. IEEE Trans Reliab. 2020;69(1):188–202. https://doi.org/10.1109/TR.2019.2925415.
84. Chung WC, Lin HP, Chen SC, Jiang MF, Chung YC. JackHare: a framework for SQL to NoSQL translation using MapReduce. Autom Softw Eng. 2014;21(4):489–508. https://doi.org/10.1007/s10515-013-0135-x.
85. Ezzat S, Mohammed I, Laila M, Yehia K. Web anomaly misuse intrusion detection framework for SQL injection detection. Int J Adv Comput Sci Appl. 2012;3(3):123–9. https://doi.org/10.14569/ijacsa.2012.030321.
86. Manikanta YVN. Protecting web applications from SQL injection attacks. 2012. p. 609–613.
87. Dharam R, Shiva SG. Runtime monitoring framework for SQL injection attacks. Int J Eng Technol. 2014;6(5):392–401. https://doi.org/10.7763/IJET.2014.V6.731.
88. Chang V, Kuo YH, Ramachandran M. Cloud computing adoption framework: a security framework for business clouds. Futur Gener Comput Syst. 2016;57:24–41. https://doi.org/10.1016/j.future.2015.09.031.
89. Yassin M, Ould-Slimane H, Talhi C, Boucheneb H. SQLIIDaaS: a SQL injection intrusion detection framework as a service for SaaS providers. In: Proceedings of the 4th IEEE international conference cyber security cloud computing CSCloud 2017 3rd IEEE international conference scalable smart cloud, SSC 2017. p. 163–170. https://doi.org/10.1109/CSCloud.2017.27.
90. Arvindpdmn L. "SQLI," 박 종 명의 아 름 다 운 개 발 Since 2010.06. 2022.
91. Yiğit G, Arnavutoğlu M. SQL injection attacks detection & prevention techniques. Int J Comput Theory Eng. 2017;9(5):351–6. https://doi.org/10.7763/IJCTE.2017.V9.1165.
92. Alwan ZS, Younis MF. Detection and prevention of SQL injection attack: a survey. J Comput Commun. 2017;06(08):1–14. https://doi.org/10.4236/jcc.2014.28001.
93. Erdődi L, Sommervoll ÅÅ, Zennaro FM. Journal of information security and applications simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents. J Inf Secur Appl. 2021;61(July):102903. https://doi.org/10.1016/j.jisa.2021.102903.
94. Abdulmalik Y. An improved SQL injection attack detection model using machine learning techniques. Int J Innov Comput. 2021;11(1):53–7.
95. Fan M, Liu J, Wang W, Li H, Tian Z, Liu T. DAPASA: detecting android piggybacked apps through sensitive subgraph analysis. IEEE Trans Inf Forensics Secur. 2017;12(8):1772–85. https://doi.org/10.1109/TIFS.2017.2687880.
96. Shunmugapriya B, Paramasivan B. Protection against SQL injection attack in cloud computing. In J Eng Res Technol. 2020;9(02):502–10.
97. Varshney K, Ujjwal RL. LsSQLIDP : literature survey on SQL injection detection and prevention techniques. J Stat Manag Syst. 2019;22(2):257–69. https://doi.org/10.1080/09720510.2019.1580904.
98. Ahmad K, Karim M. A method to prevent SQL injection attack using an improved parameterized stored procedure. Int J Adv Comput Sci Appl. 2021;12(6):324–32.
99. Kareem M. Prevention of SQL injection attacks using AWS WAF. 2018. p. 47.
100. Lockhart B, Peng J, Wu W, Wang J, Wu E. Explaining inference queries with bayesian optimization. Proc VLDB Endow. 2021;14(11):2576–85. https://doi.org/10.14778/3476249.3476304.
101. Clarke J. SQL injection inference attacks—tutorial and example.
102. Mohammed S, Chaki H, Din MM. A survey on SQL injection prevention methods, vol. 9, no. 1. 2019. p. 47–54.
103. Rawat R. "SQL injection attack detection using SVM. Int J Comput Appl. 2020. https://doi.org/10.5120/5749-7043.
104. Chen Z, Guo M. Research on SQL injection detection technology based on SVM, vol. 01004. 2018. p. 1–5.
105. Banchhor A, Vaidya T. SQL injection detection using Baye's classification. p. 313–317.

106. Olalere M, et al. A Naïve Bayes based pattern recognition model for detection and categorization of structured query language injection attack, vol. 7, no. 2. 2018. p. 189–199.
107. Liu M, Chen T. DeepSQLi : deep semantic learning for testing SQL injection. p. 286–297.
108. Liu T, Qi Y, Shi L, Yan J. Locate-then-detect : real-time web attack detection via attention-based deep neural networks. 2016. p. 4725–4731
109. Volkova M, Chmelar P, Sobotka L. Machine learning blunts the needle of advanced SQL injections. MENDEL. 2019;25(1):23–30.
110. Xie XIN, Ren C, Fu Y, Xu JIE, Guo J. SQL injection detection for web applications based on elastic-pooling CNN. IEEE Access. 2019;7:151475–81. https://doi.org/10.1109/ACCESS.2019.2947527.
111. Li QI, Li W, Wang J. A SQL injection detection method based on adaptive deep forest. 2019. p. 145385–145394. https://doi.org/10.1109/ACCESS.2019.2944951.

Publisher's Note