AJBAR Vol. 3(4), August 2024: 52-61, ISSN: 2811-2881

Arid Zone Journal of Basic and Applied Research

Faculty of Science, Borno State University
Maiduguri, Nigeria

Journal homepage: https://www.azjournalbar.com

Research Article

Statebert: Enhancing Bert with State Machine for SQL Injection Detection

Aminu Tukur[1], Kabir Umar[2] and Aliyu Isah Agaie[3]
[1]Department of Computer Science, Faculty of Computing, Bayero University Kano
[2]Department of Software Engineering, Faculty of Computing, Bayero University Kano
[3]Department of information and Media Studies, Faculty of Communication, Bayero University Kano
***Corresponding author's Email**: Tukuraminu85@gmail.com, doi.org/10.55639/607.090807

ARTICLE INFO:

Keywords:
SQL injection,
State Machine,
BERT,
Vulnerabilities

ABSTRACT

SQL injection vulnerabilities continue to be a significant threat to web applications as shown in OWASP 2021 Ranking. It enables attackers to gain unauthorized access to sensitive data and potentially execute malicious code on the server. Traditional signature-based detection methods often fail to identify novel or obfuscated SQL injection attacks. This paper presents a StateBERT approach that combines the strengths of a state machine and a BERT model to enhance the detection and mitigation of SQL injection vulnerabilities. The state machine handles the structural analysis of SQL queries, while the BERT model provides advanced language understanding to identify more complex, context-dependent anomalies. The paper discusses how this enhanced approach can effectively handle various types of SQL injection vulnerabilities. By leveraging the complementary capabilities of the state machine and the BERT model. The paper also presents the Proposed experiment, expected results and future directions. The Preliminary result is promising.

Corresponding author: Aminu Tukur, **Email:** Tukuraminu85@gmail.com
Department of Computer Science, Faculty of Computing, Bayero University Kano

INTRODUCTION

Web applications are highly important in various industries as they handle sensitive information, enable key operations, and facilitate global connections. However, despite their convenience, these applications have hidden weaknesses that can be exploited by malicious individuals to steal data, disrupt operations, or compromise entire systems. In the field of cybersecurity, computer vulnerability refers to a flaw in a system that makes it susceptible to attacks. This vulnerability can also encompass weaknesses in computers, procedures, or any other aspect that exposes information security to

threats. Some Major computer vulnerabilities include bugs, easily guessable passwords, pre-infected software, lack of data encryption, OS command and SQL injections, buffer overflow, insufficient authorization measures, utilization of flawed algorithms, and URL redirects to untrusted websites (India & Sharma, 2023).

Traditional detection methods for SQL Injection vulnerabilities, such as "Signature-based" and "Pattern Matching," primarily rely on identifying common patterns or keywords used in these attacks. While this approach can be effective against known attack techniques, it may not be adequate for detecting novel attacks that exploit previously unknown vulnerabilities. Moreover, attackers constantly develop new ways to evade detection, making it difficult for predefined patterns to catch these novel SQLi attacks (Kapoor, 2023), (Fahmi Al Azhar & Harwahyu, 2023b).

Recently, machine learning was proposed as an alternative to traditional detection methods. Although it excels at detecting common SQL injection patterns like keyword matching or suspicious function calls, it struggles with entirely new attack vectors that deviate significantly from the training data (Montaruli et al., 2023a). Deep learning models, on the other hand, can potentially achieve higher accuracy in detecting complex SQL injection attempts, including zero-day attacks (previously unknown vulnerabilities). However, the deep learning models are prone to overfitting, where they perform well on training data but poorly on unseen data (ALAzzawi, 2023).

Furthermore, a transformer based model based on "Synbert" shows promise in the detection of various forms of SQL injection by leveraging both the syntax and semantic meaning of queries beyond just keywords. Still, challenges persist in the detection of unseen queries and in the detection of piggybackings and illegal/logical queries. The approach often struggles with the evolving nature of attacks, particularly those involving malicious code within seemingly legitimate queries, where an attacker crafts queries that appear syntactically correct but still exploit vulnerabilities (Lu et al., 2023). Additionally, illegal/logical queries exploit vulnerabilities in the database schema itself, making it difficult for SynBERT to distinguish between a benign and a malicious query if the structure is similar.

Therefore, there is a need for an improved SQL injection detection approach that can effectively identify both known and unknown attack patterns, adapt to evolving attack tactics, and differentiate between legitimate and malicious queries even when the structure is similar. As a result, this study focuses on creating a SQL injection detection system that enhance bert with state machine for sql injection detection. This enhancement has the advantage of tackling the issue of previously unseen vulnerabilities and evolving attacks, which has not been extensively researched. The paper discusses how this enhanced approach can effectively handle various types of SQL injection vulnerabilities. By leveraging the complementary capabilities of the state machine and the BERT model, the system can offer an improved accuracy, enhanced contextual understanding, targeted and efficient analysis, robust and adaptable detection, and increased explainability and interpretability – all of which contribute to a more effective and reliable SQL injection vulnerability detection mechanism. The paper also present, Proposed experiment, expected results and future directions. Preliminary result. The work is ongoing; therefore, the paper Present the proposed experiment and the expected result. The Preliminary result is encouraging

The remaining of this Paper is organized as follows; section II Related Work, section III proposed method, Section IV Illustrated Example, Section IV Proposed experimental,

Section VI Expected Result and Section VII is conclusion.

Related work

In this section, we discussed the following, SQL injection overview, SQL injection detection and prevention methods, Challenges in SQL injection detection.

An SQL injection (also known as SQLi) is a technique for the "injection" of SQL commands by attackers to access and manipulate databases. Using SQL code via user input that a web application (eg, web form) sends to its database server, attackers can gain access to information, which could include sensitive data or personal customer information (Añasco Loor et al., 2023). SQL injection is a common issue with database-driven websites. Given the prevalence of such websites, this flaw is easily detected and easily exploited, and any website can be subject to an SQL injection attack (Okello et al., 2023). Many vulnerabilities detection tools have been proposed that combine more maturely technologically advanced detection methods. From the perspective of technology-driven development, SQLi vulnerability detection can be divided into traditional methods and Machine learning, where traditional methods are driven by expert experience in development. In contrast, Machine learning methods are driven by data, although in recent time there is a significant interest in Vulnerability detection using Large Language Model.

An Approach that works to explores the shortcomings of the conventional methodologies based on enumerating test case libraries and suggest a Finite State Machine (SPT-FSM)-based SQLIV Penetration Test approach. The suggested method creates an FSM based on the states that correspond to various SQLIV penetration test cases, maps test case statuses and pertinent responses, and examines the established FSM's transition regularity for testing SQLIV that has a dynamic nature and states with distinct transition characteristics. The experimental findings demonstrate that by lowering False Positives (FP) and False Negatives (FN), the suggested strategy can successfully increase the accuracy of SQLIV penetration tests (Lei *et al.*, 2016). The disadvantage is that traditional approaches rely on test case library enumerating methods, which may not be able to effectively capture the dynamic nature and state transition characteristics of SQLIV vulnerabilities.

Random forest and Support Vector Machine has shown an improvement in the detection of sqli when used with Feature extraction to identify important and new features of the data. It yields improved result and speed up in the training process as well. Originally, the dataset provided a feature and a class attribute which are "length", and "attack_type". The Random Forest as shown has a significant benefit in that it can be used for both classification and regression issues (Dass, 2022). However, if the training data is incomplete, contains errors, or lacks representative samples of SQL injection attacks, the model may not perform well in detecting such attacks. Also several hyperparameters need to be tuned for optimal performance. Choosing the right values for these parameters can be challenging and might require substantial computational resources and time. If not appropriately tuned, models can exhibit decreased performance.

A hybrid framework for detecting structured query language injection attacks in web-based applications that combines a SQL query matching technique (SQLMT) and a standard blockchain framework to detect SQLi attacks created by insiders. In contrast to alternative methods, this suggested framework is relatively easy to put into practice. However, it does require some extra computational time because it relies on confirmations from peers working on the blockchain system. Nevertheless, the increased computational cost is justified as the process of checking query integrity provides a

more efficient solution for detecting SQL injection attacks (Furhad et al., 2022). The limitation is that the approach utilizes a SQL query matching technique (SQLMT) to detect malicious SQL queries. This will assumes that the system has a pre-defined set of "normal" SQL query patterns, which may not always be comprehensive or up-to-date.

In another study, Natural Language Processing (NLP) was utilized to improve the accuracy of text processing for detection. Four algorithms were implemented including Logistic Regression, Naive Bayes, Random Forest, and Convolutional Neural Network (CNN). The model was trained using data from the first dataset and evaluated using validation records from the second dataset. The primary dataset was transformed into a corpus to facilitate the model's understanding of the input. The CNN algorithm demonstrated the highest accuracy, and faster response time. Compared to traditional machine learning method (Natarajan *et al.,* 2023). Although deep learning has outperformed the machine learning models, deep learning model is prone to overfitting.

Furthermore, a model that uses supervised machine learning techniques, input string validation technique, and pattern matching to identify anomalies-injections has shown a significant improvement over other machine learning model like KNN and RF (Irungu et al., 2023). The proposed model was not tested against other common SQL injection attack types, such as union-based, blind, and error-based attacks. Evaluating the model's performance on a wider range of SQL injection techniques would provide a more comprehensive assessment of its capabilities; it was tested on the tautology attack.

Additionally, the issue of explicability and interpretability of black-box models was introduced and provide insights into the model's decision-making process using a case-based explanation method for classifying SQL sentences as an attack or not(Recio-Garcia et al., 2023). Also, another Approach emphasizes on the importance of protecting user data and privacy in the age of technology and the need to increase website security to prevent cyber-attacks. The major findings of their study conclude that the naive Bayes approach showed significant accuracy in identifying SQL injection threats with 0.99 accuracies. Naive Bayes Performs better than Neural-Network, SVM, Random-Forest, KNN, and Logistic Regression (ALMaliki & Jasim, 2023). The method does not mention any external validation or testing of the models on a separate dataset or in a different context.

A model called "SQIRL" is used for Grey-Box Detection Using Reinforcement Learning. Their model generates a more diverse range of potential risks compared to existing scanners, resulting in the discovery of more vulnerability. Additionally, SQIRL employs a targeted approach, leading to fewer payloads being attempted. In testing against a microbenchmark for SQL injection, SQIRL successfully identifies all vulnerabilities with significantly fewer requests than most state-of-the-art scanners. Furthermore, in a separate evaluation involving 14 real-world web applications, SQIRL outperforms other scanners by uncovering 33 vulnerabilities without any false positives. The researchers who developed SQIRL responsibly disclosed 22 newly discovered vulnerabilities, which have been categorized into 6 CVEs (Wahaibi et al., 2023). The approach is limited by their inefficient payload generation techniques, which rely on simple rule-based approaches that cover only common payloads and fail to adapt to the specific characteristics of target web applications, leading to a high volume of ineffective requests. Logistic Regression-based model on the hand demonstrates that detecting SQL injection attacks on flow data from lightweight protocols is possible, with a false alarm rate of less than

0.07% with a (Crespo-Martínez et al., 2023). The technique relies solely on network flow data (e.g., NetFlow, sFlow) and does not utilize the full packet-level information that may be available in some network environments

Contextualized word embeddings increase the precision rate of machine learning-based SQL injection attack detection to over 99% over a variety of classification algorithms. Additionally, the model's training period is shortened by a noteworthy factor of 31. Furthermore, contextualized embeddings allow better model calibration, as shown by reliability diagrams (Zulu et al., 2024). While the technique demonstrates superior performance of contextualized embeddings in a research setting, the practical deployment and real-world applicability of the approach are not discussed. Factors such as computational overhead, integration with existing systems, and monitoring for evolving attack patterns may need further Analysis. Another approach that creates multiple perspectives of SQL data by encoding SQL statements into SQL tags. The method involves using bidirectional long short-term memory (LSTM) and convolutional neural network (CNN) layers to learn a shared latent space from these different views. In the detection stage, the method makes individual predictions for each representation and determines if a query is an SQLi attack by considering the output of a consensus function. Furthermore, the method uses an approach called Interpretable Model-Agnostic Annotations (LIME) to interpret the results of the model and analyze the SQL injection inputs. This is part of the Explainable Artificial Intelligence (XAI) field. The results show that the MVC-BiCNN method performed better than baseline methods, achieving a detection rate of 99.96% (Kakisim, 2024). The use of a multi-view approach with bidirectional LSTM and CNN layers may increase the computational complexity of the model, which could impact the real-time processing and deployment of the system.

Despite the continuous efforts of researchers, the problem of SQL injection (SQLi) continues to pose a significant threat in the digital landscape. It remains a threat within web applications, constantly evolving its attack techniques to exploit vulnerabilities. Although there are various detection techniques available, the fight against SQLi is far from being won. This ongoing battle is accompanied by several challenges, including the presence of false positives and negatives in detection, the complexity of web application logic, concerns regarding data security, and the limitations of available resources.

Therefore, this research introduces a new approach called StateBERT, The system would enhance the detection of SQL injection, leveraging the complementary strengths of the state machine and BERT components. The operation of the model is showed in section III.

METHODOLOGY

This chapter presents the general research methodology that will be used in conducting this study, shown as a sequence of activities which will be carried out in phases and steps.

This research proposed a model for the detection of SQL injection vulnerability by integrating State Machine and BERT. The model is Enhanced State machine and Bert model called **StateBert.** The Left side of the diagram is the state machine while the right side is the BERT part of the model It utilize both features of State machine and BERT as shown in Figure 1.

A state machine will provide a structured representation of the expected behavior and valid transitions in an application's input processing flow. The State Machine accepts an SQL injection query as input, a state will analyze the query, which will define the legitimate sequence of states that an application's inputs should follow. It also identifies anomalous or malicious input patterns that deviate from the expected flow. The state

machine will capture the valid structure and syntax of SQL queries, allowing the model to detect deviations or injections that violate the expected patterns.

The state machine transitions between states based on the character being processed in the input string, e.g Valid Character. The machine transitions to the 'Error' state (invalid) if an unexpected character or sequence is encountered, indicating a potential SQL injection attempt. The state machine accepts the input as valid if it reaches a designated end state (e.g., a state representing a complete SQL statement) without transitioning to the Error state. The state machine rejects the input as potentially containing SQL injection if it transitions to the 'Error' state at any point during processing. Finally, the acceptance state will be the output; it will consist of both malicious and benign queries.

Furthermore, The BERT model will convert the state machine output into a sequence of token IDs, which represent the individual tokens (words, symbols, etc.) present in the output. This can be achieved by mapping each unique state and metadata element (e.g., SELECT, FROM, WHERE, CONDITION) to a unique token ID, creating a vocabulary of the state machine output elements.

The sequence of states and metadata elements from the state machine output is then converted to a sequence of token IDs, which can be used as the input to the BERT model.

The encoded state machine output, represented as a sequence of token IDs, is then passed as input to the BERT-based detection mechanism. The BERT model takes the input sequence and processes it through its multi-layer transformer architecture, generating a contextual representation of the input.The final layer of the BERT model produces a classification output, which indicates whether the input SQL query is classified as "normal" or "malicious." The BERT model's classification is based on its learned patterns and characteristics from the pre-training dataset, which includes both benign and malicious SQL queries. If the BERT model classifies the input as "malicious," it suggests that the state machine output contains anomalies or suspicious patterns that are indicative of a potential SQL injection attack.

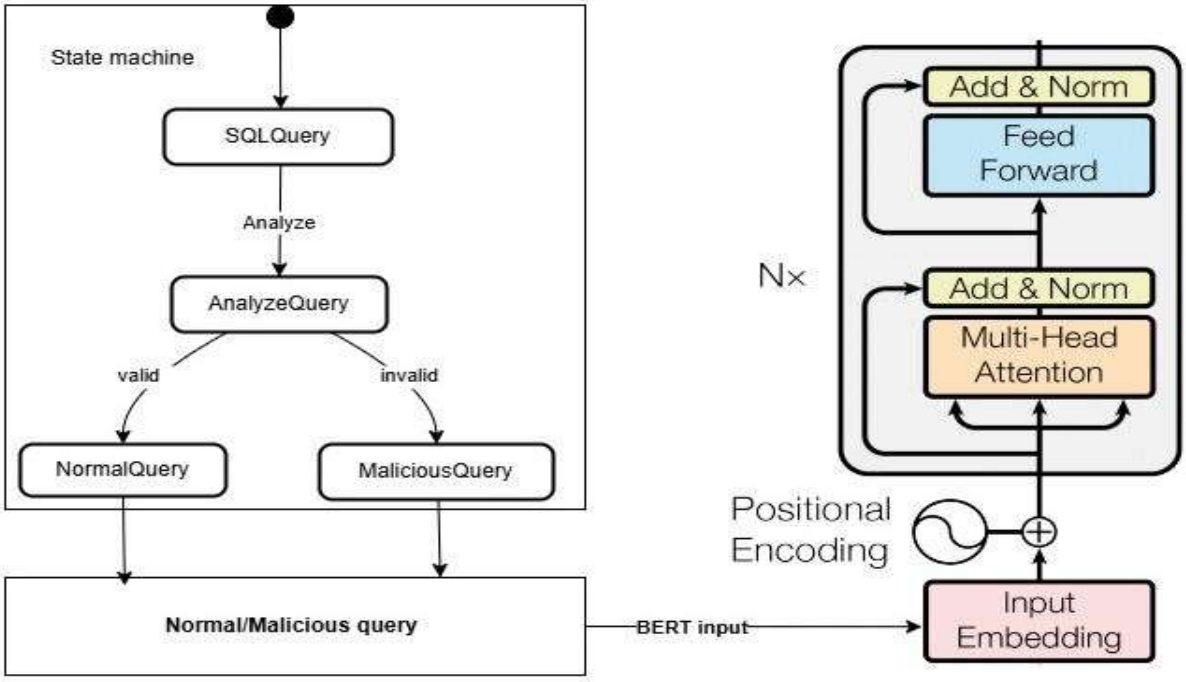

Figure 1: Model of the Proposed StateBERT

By integrating the state machine output as the input to the BERT-based detection mechanism, the SQL injection detection system can leverage the complementary strengths of both approaches to enhance the accuracy and reliability of SQL injection detection. The state machine's structural analysis provides valuable information to the BERT model, which can then apply its advanced language understanding capabilities to make a more informed decision about the nature of the SQL query.

Finally, evaluation and validation of the performance of the integrated approach using appropriate metrics and testing methodologies. The experiment to be carried out is shown is in section IV, V and VI bellow.

Illustrated Example

In this section, a real example is used to demonstrate what is expected from the system.

Example 1: Normal SQL Query

```
SELECT * FROM users
WHERE id = 1
```

The state machine would process the SQL query and identify the following key components:

[{'state': 'SELECT', 'start': 0, 'end': 6}, {'state': 'FROM', 'start': 7, 'end': 11}, {'state': 'WHERE', 'start': 12, 'end': 19}, {'state': 'CONDITION', 'start': 20, 'end': 25}]

'start': This represents the starting position of the identified component or clause within the original SQL query string. In the example, the 'start' is 0, which means the SELECT clause starts at the 0th (first) character position of the SQL query.

'end': This represents the ending position of the identified component or clause within the original SQL query string. In the example, the 'end' is 6, which means the SELECT clause ends at the 6th character position of the SQL query.

The state machine would recognize the SELECT, FROM, WHERE,
and CONDITION clauses, and capture their respective positions within the query. The structured output from the state machine would be encoded as a sequence of token IDs and fed into the BERT model. The BERT model would analyze the input and look for any patterns or anomalies that might indicate a SQL injection vulnerability. In this case, the BERT model would classify the SQL query as "normal," as it does not contain any suspicious or malicious elements.

The state machine's structural analysis did not identify any anomalies, and the BERT model's classification of the query as "normal" indicates that the SQL query is safe and does not contain any SQL injection vulnerabilities. The combined detection mechanism would provide a high-confidence assessment that the SQL query is safe and can be executed without any security concerns.

Example 2: Malicious SQL Query

```
SELECT * FROM users WHERE id = 1
OR '1'='1'
```

The state machine would process the SQL query and identify the following key components:

[{'state': 'SELECT', 'start': 0, 'end': 6}, {'state': 'FROM', 'start': 7, 'end': 11}, {'state': 'WHERE', 'start': 12, 'end': 19}, {'state': 'CONDITION', 'start': 20, 'end': 32}]

The state machine would recognize the SELECT, FROM, WHERE, and the malicious CONDITION clause, which includes the suspicious OR '1'='1' condition. The structured output from the state machine would be encoded as a sequence of token IDs and fed into the BERT model. The BERT model would analyze the input and look for any patterns or anomalies that might indicate a SQL injection vulnerability. In this case, the BERT model would classify the SQL query as "malicious," as it contains a suspicious condition that is characteristic of a SQL injection attack.

The state machine's structural analysis identified the malicious CONDITION clause, and the BERT model's classification of the query as "malicious" confirms that the SQL query

contains SQL injection vulnerability. The combined detection mechanism would provide a high-confidence assessment that the SQL query is malicious and should not be executed, as it poses a significant security risk.

In the first example, both the state machine and the BERT model recognized the SQL query as normal, indicating that it is safe and does not contain any SQL injection vulnerabilities. In the second example, the state machine identified the malicious CONDITION clause, and the BERT model confirmed that the query is malicious, effectively detecting the SQL injection vulnerability.

Proposed Experiment

In this section, the set of experiment to be carried out is described. It contains 3 different components; Performance metrics, Data sets, and Expected result

PERFORMANCE METRICS FOR SQL INJECTION EVALUATION

Protecting against SQL injection (SQLi) is crucial in today's digital landscape. Although most studies use similar evaluation metrics; Accuracy, Precision, Recall and F1-score. Here's a set of evaluation metrics to assess the SQLi detection and prevention strategies. Most studies use similar evaluation metrics:

- $\boldsymbol{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$ equ 1
- **Precision** = $\frac{TP}{TP+FP}$ equ 2
- $\boldsymbol{Recall} = \frac{TP}{TP+FN}$ equ 3
- $\boldsymbol{F1\ Score} = \frac{2TP}{2TP+FP+FN}$ equ 4

Where;

- **True Positive (TP):** Measures the percentage of actual SQLi attacks correctly identified. Higher TPR indicates better detection accuracy.
- **False Positive (FP):** Measures the percentage of legitimate requests flagged as malicious. Lower FPR minimizes disruptions to legitimate users.
- **True Negative (TN):** Measures the percentage of legitimate requests correctly identified as safe. Higher TN signifies strong baseline protection.
- **False Negative (FN):** Measures the percentage of missed SQLi attacks. Lower FN indicates robust detection capabilities.

DATASET

The dataset that will contain **benign SQL query** and **malicious Query** from "Kaggle" SQl This dataset is composed of 30,920 samples of SQL queries . Thus, it contains 11,331 Malicious and 19,589 benign queries. For testing it will be selected randomly from the original dataset, where 2,000 are benign queries, while the other 2,000 are SQLi queries.

Table 5.1: Kaggle SQL Dataset

S/No	Dataset	Malicious	Normal	Total
1.	Kaggle	11331	19589	30920

EXPERIMENTAL PLANNING

In this study, After Successful implementation of StateBERT model, N-Fold Validation will be carried out, We will use the KFold class from the sklearn.model_selection module to perform the k-fold cross-validation. The cross_validate_sql_injection_detector function takes the input data (X) and labels (y) as arguments, and the number of folds (k) to use.

The function will splits the data into k folds, trains the SQL injection detection model on k-1 folds, and evaluates it on the remaining fold. This process is repeated k times, and the mean performance metrics (accuracy, precision, recall, F1-score) are calculated and returned.

By using cross-validation, we can better estimate the model's performance and its ability to generalize to unseen SQL injection patterns. This information can help make more informed decisions about the model's deployment, identify areas for improvement, and track the model's performance over time as the SQL injection landscape evolves.

CONCLUSION AND FUTURE WORK

The research presented in this paper introduces a StateBERT approach that combines the strengths of a state machine and a BERT model to enhance the detection and mitigation of SQL injection vulnerabilities. This system represents a promising alternative and potential improvement over existing techniques and its limitations in their ability to comprehensively handle the diverse range of SQL injection attack vectors. The proposed approach aims to overcome these limitations by leveraging the complementary capabilities of the state machine and the BERT model. The state machine's structural analysis of SQL queries allows for the effective identification of syntax-based SQL injection attempts, while the BERT model's advanced language understanding enables the detection of more complex, context-dependent anomalies, such as tautology-based, union-based, and inference-based SQL injection vulnerabilities. This combination of techniques provides a more holistic and robust defense against SQL injection threats. As the research progresses, the continued evaluation and refinement of the approach, including its performance, scalability, and real-world applicability, is crucial and will be Presented. Ongoing collaboration with security practitioners and the integration of the hybrid system into existing security frameworks may also yield insights and drive further enhancements.

REFERENCES

ALMaliki, M., & Jasim, M. (2023). Comparison study for NLP using machine learning techniques to detecting SQL injection vulnerabilities. *International Journal of Nonlinear Analysis and Applications*, *14*(8). https://doi.org/10.22075/ijnaa.2022.28365.4098

Añasco Loor, C., Morocho, K., & Hallo, M. (2023). Using Data Mining Techniques for the Detection of SQL Injection Attacks on Database Systems. *Revista Politécnica*, *51*(2), 19–28. https://doi.org/10.33333/rp.vol51n2.02

Arasteh, B., Aghaei, B., Farzad, B., Arasteh, K., Kiani, F., & Torkamanian-Afshar, M. (2024). Detecting SQL injection attacks by binary gray wolf optimizer and machine learning algorithms. *Neural Computing and Applications*, *36*(12), 6771–6792. https://doi.org/10.1007/s00521-024-09429-z

Crespo-Martínez, I. S., Campazas-Vega, A., Guerrero-Higueras, Á. M., Riego-DelCastillo, V., Álvarez-Aparicio, C., & Fernández-Llamas, C. (2023). SQL injection attack detection in network flow data. *Computers & Security*, *127*, 103093. https://doi.org/10.1016/j.cose.202103093

Dass, F. D. M. (2022). *A Comparative Study of SQL Injection Detection Using Machine Learning Approach*. *3*(2).

Falor, A., Hirani, M., Vedant, H., Mehta, P., & Krishnan, D. (2022). A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks. In D. Gupta, Z. Polkowski, A. Khanna, S. Bhattacharyya, & O. Castillo (Eds.), *Proceedings of Data Analytics and Management* (Vol. 91, pp. 293–304).

Springer Singapore. https://doi.org/10.1007/978-981-16-6285-0_24

Furhad, Md. H., Chakrabortty, R. K., Ryan, M. J., Uddin, J., & Sarker, I. H. (2022). A hybrid framework for detecting structured query language injection attacks in web-based applications. *International Journal of Electrical and Computer Engineering (IJECE)*, *12*(5), 5405. https://doi.org/10.11591/ijece.v12i5.pp5405-5414

Harip, S. H. N., Hamid, I. R. A., Murli, N., & Hassan, N. (2022). *Classification of SQL injection attack using K-Means clustering algorithm*. 040004. https://doi.org/10.1063/5.0104348

Irungu, J., Graham, S., Girma, A., & Kacem, T. (2023). Artificial Intelligence Techniques for SQL Injection Attack Detection. *Proceedings of the 2023 8th International Conference on Intelligent Information Technology*, 38–45. https://doi.org/10.1145/3591569.3591576

Kakisim, A. G. (2024). A deep learning approach based on multi-view consensus for SQL injection detection. *International Journal of Information Security*, *23*(2), 1541–1556. https://doi.org/10.1007/s10207-023-00791-y

Kapoor, A. (2023). SQL-Injection Threat Analysis and Evaluation. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4430812

Lei, L., Xu, Jing, Guo, Ghenkai, Kang, Jiehui, Xu, Sihan, & Zhang, Biao. (2016). Exposing SQL Injection Vulnerability through Penetration Test based on Finite State Machine. *2nd IEEE International Conference on Computer and Communications (ICCC)*, 1171–1175. https://doi.org/10.1109/CompComm.2016.7924889

Lu, D., Fei, J., & Liu, L. (2023). *A Semantic Learning-Based SQL Injection Attack Detection Technology*.

Natarajan, Y., Karthikeyan, B., Wadhwa, G., Srinivasan, S. A., & Akilesh, A. S. P. (2023). A Deep Learning Based Natural Language Processing Approach for Detecting SQL Injection Attack. In A. Abraham, S. Pllana, G. Casalino, K. Ma, & A. Bajaj (Eds.), *Intelligent Systems Design and Applications* (pp. 396–406). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-35507-3_38

Recio-Garcia, J. A., Orozco-del-Castillo, M. G., & Soladrero, J. A. (2023). *Case-based Explanation of Classification Models for the Detection of SQL Injection Attacks*.

Singh, K. D., & Singh, P. D. (2023). Machine Learning in Robotics with Fog/Cloud Computing and IoT. *EAI Endorsed Transactions on AI and Robotics*, *2*. https://doi.org/10.4108/airo.3621

Sun, H., Du, Y., & Li, Q. (2023). Deep Learning-Based Detection Technology for SQL Injection Research and Implementation. *Applied Sciences*, *13*(16), 9466. https://doi.org/10.3390/app13169466

Tadhani, J. R., Vekariya, V., Sorathiya, V., Alshathri, S., & El-Shafai, W. (2024). Securing web applications against XSS and SQLi attacks using a novel deep learning approach. *Scientific Reports*, *14*(1), 1803. https://doi.org/10.1038/s41598-023-48845-4

Venkatramulu, S., Waseem, S., Taneem, A., Thoutam, S. Y., & Apuri, S. (2024). *Research on SQL Injection Attacks using Word Embedding Techniques and Machine Learning*.

Zulu, J., Han, B., Alsmadi, I., & Liang, G. (2024). Enhancing Machine Learning Based SQL Injection Detection Using Contextualized Word Embedding. *Proceedings of the 2024 ACM Southeast Conference on ZZZ*, 211–216. https://doi.org/10.1145/3603287.3651187