VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SQL INJECTION PREVENTION AND DETECTION

Course: Information Security Analysis and Audit
Course code: BCSE353E
Faculty: Abirami S
Slot: TB2

SUBMITTED BY:
Runi Gosh (21BCE5217)
Rupkatha De (21BCE6075)
Shreyash Belgaonkar (21BCE6107)
Pranjal Tiwari (21BCE6198)

LITERATURE REVIEW

Research Paper 1:-

Title : SQL Injection Prevention by Adaptive Algorithm

Author : Ashish John - Dept. of Computer Science and Engineering,SRM University, NCR Campus

The paper addresses the critical issue of SQL injection attacks, which pose significant security threats to web applications. This literature review highlights the existing research and techniques related to SQL injection prevention. The authors emphasises that SQL injections provide an open door for hackers to compromise web interfaces and manipulate databases. They stress that preventing such attacks is the responsibility of web developers rather than web hosting providers.

The study focuses on the login phase and proposes a novel solution that combines the Parse Tree Validation Technique and Code Conversion Method. It highlights the importance of addressing SQL injection vulnerabilities due to the sensitive and confidential information stored in back-end databases. It identifies various types of SQL injection attacks, including tautology attacks, piggy-backed queries, union queries, illegal/logically incorrect queries, and stored procedures. Each attack type is explained with examples to illustrate their potential impact on the system.

Several existing techniques are mentioned in the review, such as parse tree validation and code conversion. The parse tree validation technique involves comparing the parse tree of the SQL statement before and after the inclusion of user input to detect vulnerabilities. On the other hand, code conversion converts user input into different representations (e.g., ASCII, binary, hex) and checks its availability in the data table.

The review concludes that a combination of these techniques is essential to provide robust protection against SQL injection attacks. It acknowledges the limitations of individual methods and emphasises the need for continuous monitoring and improvement of security measures. The authors also highlight the trade-off between security and efficiency, as excessive code conversion can increase processing time and database size.

In summary, the literature review underscores the significance of SQL injection prevention and presents an overview of different attack types and existing techniques. The proposed adaptive algorithm offers a promising approach to enhance the security of web applications during the login phase. However, further research is needed to evaluate its effectiveness, scalability, and applicability in real-world scenarios.

Research Paper 2:-

Title : Prevention Of Sql Injection Attack Using Unsupervised MachineLearning Approach

Author :

- **M.N.Kavitha - Assistant Professor, Department Of Computer Technology (Ug), Kongu Engineering College, Tamil Nadu, India.**
- **V. Vennila - Associate Professor, Department Of Computer Science & Engineering, K.S.R. College Of Engineering, Tamil Nadu, India**
- **G. Padmapriya - Associate Professor, Department Of Computer Science & Engineering, Saveetha School Of Engineering, SaveethaInstitute Of Medical And Technical Sciences, Tamil Nadu, India**
- **A. Rajiv Kannan - Professor and Head, Department Of Computer Science & Engineering, K.S.R. College Of Engineering, Tamil Nadu**

The paper titled "Prevention of SQL Injection Attack Using Unsupervised Machine Learning Approach" discusses the use of machine learning techniques, specifically unsupervised learning, to prevent SQL injection attacks in web applications. The authors propose a system that incorporates machine learning with a web application firewall (WAF) to enhance its effectiveness in detecting and preventing SQL injection attacks.

The paper begins by highlighting the increasing exposure of online web applications to various types of attacks, particularly SQL injection attacks, where attackers modify SQL queries to gain unauthorised access to and manipulate information in the web application or database. The authors emphasise the need for regular updating and testing of WAFs to prevent such attacks. They propose the use of unsupervised machine learning, specifically the K-means clustering algorithm, as an approach to enhance the security of existing systems.

The proposed system follows a flow where the end user makes a query in the web application, and the values of the query are extracted and sent to the SQL injection detector. The system provides two layers of security: the first layer uses context-free grammar (CFG) to create patterns for low-level attacks, while the second layer utilises unsupervised learning to train and detect high-level attacks. The authors also discuss the scope and application of the proposed methodology, highlighting its potential for adapting to different web application scenarios.

The paper includes a literature review section that summarises related works in the field of firewalls and SQL injection detection. It mentions research on integrating machine learning with evolutionary algorithms, search-based approaches, and various testing techniques for SQL injection detection. The authors also highlight the limitations of existing approaches and the need for more effective and automated detection methods.

In the methodology section, the authors provide a detailed explanation of the proposed system architecture, functional architecture, and three main steps of the methodology : URL intercept engine, context-free grammar for SQL injection attacks and pattern classification through machine learning. They describe how the system intercepts, checks URL values, creates attack patterns using CFG, and applies machine learning algorithms for pattern classification.

The paper concludes by summarising the efforts undertaken in the proposed system and discussing its findings and limitations. It highlights the potential of using unsupervised machine learning for SQL injection attack prevention and suggests avenues for future research.

In summary, the paper presents a novel approach to preventing SQL injection attacks using unsupervised machine learning. It discusses the system architecture, methodology, and the potential benefits of incorporating machine learning techniques into existing web application firewalls. The proposed approach shows promise in enhancing the security of web applications and protecting against SQL injection attacks.

Research Paper 3:-

Title : SQL Injection Attacks Predictive Analytics Using SupervisedMachine Learning Techniques

Author :

- **Akinsola, Jide E. T. Department of Computer Science, Babcock University, Ilisan-Remo, Ogun State, Nigeria**
- **Awodele, Oludele Department of Computer Science, Babcock University, Ilisan-Remo, Ogun State, Nigeria**
- **Idowu, Sunday A. Department of Computer Science, Babcock University, Ilisan-Remo, Ogun State, Nigeria**
- **Kuyoro, Shade O. Department of Computer Science Babcock University, Ilisan-Remo, Ogun State**

This research paper focuses on the detection and prevention of Structured Query Language Injection Attacks (SQLIA) in web applications using machine learning techniques. SQLIA is a common cyber attack that exploits vulnerabilities in web applications to gain unauthorised access, manipulate data, and bypass authentication mechanisms.

The paper explores different approaches for mitigating SQLIA, including cryptography, XML, pattern matching, parsing, and machine learning. The machine learning approach, implemented through defensive coding, has shown promising results in SQLIA mitigation. The paper conducts an experimental analysis using various supervised learning classification algorithms, such as Logistic Regression, Stochastic Gradient Descent, Sequential Minimal Optimization, Bayes Network, Instance-Based Learner, Multilayer Perceptron, Naive Bayes, and J48.

The performance of these algorithms is evaluated using Hold-Out and 10-fold Cross Validation techniques, considering metrics like accuracy and time to build the model. The results show that Instance-Based Learner (IBK) performs the best in terms of accuracy, sensitivity, specificity, and time to build the model. The study emphasises the importance of considering multiple performance evaluation metrics for optimal algorithm selection in predictive analytics for cyber security applications.

Machine learning is a specialised area of artificial intelligence that focuses on enabling computers to learn from data and perform tasks automatically. It is widely used in various fields such as healthcare, science, engineering, business, and finance. There are several types of machine learning techniques, including supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning, evolutionary learning, and deep learning.

Supervised learning trains a system using known input and output data to predict future outputs. It is commonly used for prediction and classification tasks. On the other hand, unsupervised learning aims to find underlying data structures and hidden patterns without labeled responses. Clustering is a prevalent unsupervised method used for grouping data. Semi-supervised learning lies between supervised and unsupervised techniques, utilising a combination of labeled and unlabelled training data.

Reinforcement learning focuses on how software agents should take actions in an environment to optimise rewards. It is commonly used in areas like gaming and robotics. Evolutionary learning is a subset of evolutionary computation that utilises biological evolutionary processes to solve optimisation problems. Deep learning, a part of machine learning, relies on data representations instead of specific algorithms for tasks. It is widely applicable in computer vision, natural language processing, voice and sound identification, and various other domains.

Machine learning algorithms can be classified into different categories. Function classifiers, such as logistic regression, Sequential Minimal Optimisation (SMO), and Multilayer Perceptron (MLP), classify data based on mathematical functions. Bayes classifiers, including Bayes Network (BNK) and Naive Bayes (NBS), use probabilistic techniques based on Bayes' Theorem. Tree classifiers, like J48, use decision tree algorithms to determine the behaviour of attributes in instances. Lazy classifiers, such as Instance Based Learner (IBK), are based on the k-Nearest Neighbours (k-NN) algorithm, which classifies data based on clusters. These algorithms are evaluated based on performance metrics like accuracy, sensitivity, specificity, Kappa statistic, and training time.

In the experimental analysis, the WEKA software was used to evaluate the performance of the machine learning algorithms. Hold-out and 10-fold cross-validation methods were employed to assess the algorithms' performance and select the best ones. The evaluation considered various performance metrics, including accuracy, sensitivity, specificity, Kappa statistic, and training time.

The results showed that different algorithms performed differently based on the specific metrics evaluated. For example, SDG, J48, and LRN performed well in terms of accuracy in the hold-out method, while J48, SMO, and SDG were the top performers in 10-fold cross-validation. Sensitivity values were similar for several algorithms, making it challenging to choose the best classifier solely based on this metric. Specificity results were also consistent for multiple algorithms, except for LRN, BNK, and NBS. The Kappa statistic indicated high similarity between ensemble members for SDG and J48.

In conclusion, the selection of the optimal machine learning algorithm depends on the specific performance metrics and evaluation methods used. Different algorithms may excel in different metrics, and it is essential to consider multiple criteria for choosing the most suitable algorithm for a given task.

Research Paper 4:-

Title : Review of SQL Injection Attack and Proposed Method for Detection and Prevention of SQLIA

Author : Mayank Namdev ; Fehreen Hasan ; Gaurav Shrivastav

Department Of Computer Science and Engineering, R.K.D.F. Instituteof Science and Technology, Bhopal (M.P.) India

The paper provides an overview of SQL Injection attacks (SQLIA) and discusses various methods to prevent them. The authors emphasise the importance of information security in today's digital landscape and highlight the increasing number of SQLIA incidents reported in recent years.

The paper begins by explaining the concept of SQL Injection, which is a web application security vulnerability. It occurs when an attacker is able to submit a malicious SQL command to a web application, leading to unauthorised access to the back-end database. The attacker exploits weaknesses in input validation or encoding, allowing them to execute unintended commands or manipulate data stored in the database. SQL Injection attacks can have severe consequences, such as financial fraud, data theft, website defacement, and sabotage.

To prevent SQLIA, the paper explores different models and techniques that have been proposed. One common approach is to use database stored procedures instead of direct SQL statements in web applications. Stored procedures use parameterised queries, which are less prone to SQLIA. However, vulnerabilities can still exist when dynamic SQL statements are used within stored procedures. The paper focuses on addressing this specific vulnerability and presents a detection scheme for SQLIA in scenarios involving dynamic SQL statements.

The authors classify SQLIA into different types, including tautologies, illegal/ logically incorrect queries, piggy-backed queries, and inference-based attacks. They provide examples and explanations of each type to demonstrate how attackers exploit these vulnerabilities. The paper also discusses related work in the field, highlighting various techniques such as static analysis, runtime monitoring, prepared statement generation, automatic test case generation, and hash value approaches that have been proposed to detect and prevent SQLIA.

In conclusion, the paper provides an overview of SQL Injection attacks, their potential impact, and the importance of preventing them. It explores different types of SQLIA and discusses proposed methods and techniques for detection and prevention. The authors also present their own scheme for handling SQLIA in scenarios involving dynamic SQL statements. The paper contributes to the ongoing research and development efforts aimed at improving web application security and safeguarding against SQL Injection attacks.

Research Paper 5:-

Title :
A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm.

Author : Oluwakemi Christiana Abikoye , Abdullahi Abubakar , Ahmed Haruna Dokoro , Oluwatobi Noah Akande and Aderonke Anthonia Kayode.

Inference:

This paper discusses the increasing popularity of web applications and services, as well as the growing concern regarding web security. With the rise of the internet and the number of users accessing web applications, hackers have targeted vulnerabilities within these applications, such as SQL injection and cross-site scripting (XSS) attacks.

SQL injection is a type of attack where hackers manipulate user-supplied data in SQL queries to gain unauthorized access to or manipulate a database. By inserting SQL meta-characters into input fields, hackers can alter the structure of a query and bypass authentication mechanisms. For example, by injecting code like "'anything' OR '1' = '1'; #", the hacker can manipulate the WHERE clause of a query to gain access without knowing the password.

XSS attacks, on the other hand, involve hackers injecting malicious JavaScript code into trusted websites, which can then be executed by users' browsers. This allows the attacker to redirect users to malicious websites, steal user sessions, or gather sensitive information like login credentials. XSS vulnerabilities can be categorized as reflected, stored, or DOM-based, depending on how the attack is executed.

The paper mentions that SQL injection and XSS attacks are considered the most dangerous among the various web application vulnerabilities. Various prevention techniques have been proposed in the literature, including data encryption algorithms, pattern matching algorithms, PHP escaping functions, and instruction set randomization. However, no single technique can address all categories of vulnerabilities, and new techniques for detecting and preventing these attacks are continuously needed.

Overall, the paper highlights the significance of web security in the context of the increasing reliance on web applications and services. It emphasizes the need for effective measures to prevent SQL injection and XSS attacks, given their potential to compromise user data and the criticality of the affected applications.

Next, the research paper discusses a proposed technique to detect and prevent SQL injection and XSS attacks. The paper describes the development of a filter function using the KMP string matching algorithm to identify attack patterns. The filter function is designed to check for various forms of SQL injection attacks (such as Boolean-based, union-based, error-based, batch query, and like-based) as well as XSS attacks. If any of the functions within the filter function detect an attack pattern, the user is blocked, the HTTP request is reset, and a warning message is displayed.

The paper discusses the implementation of the proposed technique using PHP scripting language and Apache XAMPP server. The technique is tested in different environments, including various operating systems and web browsers. A vulnerable web application is purposely developed to test the effectiveness of the technique. The results indicate that the proposed technique successfully detects and prevents all tested forms of SQL injection and XSS attacks.
The paper compares the proposed technique with existing methods, including those using pattern matching algorithms, data encryption algorithms, instruction set randomization, and PHP escaping functions. The comparison reveals that the proposed technique outperforms existing methods by effectively detecting and preventing different forms of SQL injection and XSS attacks, including encoded injection attacks.
In conclusion, the proposed technique presents a novel approach to mitigate SQL injection and XSS attacks by utilizing a filter function based on the KMP string matching algorithm. The technique demonstrates a high level of effectiveness in detecting and preventing

various attack patterns, making it a valuable contribution to web security.

Research Paper 6:-

Title : A Literature Review and Comparative Analyses on SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques

Author : Bojken Shehu and Aleksander Xhuvani

Inference:

The paper discusses the importance of information security in web applications and highlights SQL injection attacks as one of the major threats to web application security. It describes various types of vulnerabilities and SQL injection attacks that can be exploited by hackers. Here is a summary of the main points:

Internet and Information Security:

The internet is a vast information infrastructure, and information and data have become crucial business assets. However, the internet is also vulnerable to security threats, such as SQL injection attacks, which can lead to financial fraud, data theft, and other malicious activities.

SQL Injection Background:

SQL injection is a web attack mechanism used by hackers to steal data from organizations. It allows attackers to add malicious SQL code to input fields of a web form to gain unauthorized access or make changes to the underlying database.

Types of Vulnerabilities:

The text mentions three types of vulnerabilities commonly found in web programming languages: input validation, lack of clear data type distinction, and delay of operations analysis. These vulnerabilities can be exploited by attackers to execute malicious code.

Types of SQL Injection Attacks:

The text describes several types of SQL injection attacks, including tautologies, logically incorrect queries, union queries, stored procedures, piggy-backed queries, inference attacks (such as blind injection and timing attacks), and alternate encodings. Each attack aims to manipulate the SQL queries executed by the application to gain unauthorized access, extract data, or perform other malicious actions.

Related Work:

The text briefly mentions some existing techniques and tools for detecting and preventing SQL injection attacks. One example is WAVES, a black-box testing technique that uses a web crawler and machine learning to identify SQL injection vulnerabilities in web applications. Another example is WebSSARI, which uses static analysis to check taint flows against SQL injection vulnerabilities.

Overall, it emphasizes the significance of addressing SQL injection attacks and the need for robust security measures to protect web applications from such threats. It acknowledges that while various tools and techniques exist for detecting and preventing SQL injection, achieving absolute security is still challenging.

The literature review presented in this section compares various SQL injection detection and prevention techniques. The review focuses on analyzing the capabilities of these techniques in detecting and preventing different types of SQL injection attacks. The comparison is based on the findings from several articles, rather than empirical experience.

For SQL injection detection techniques, summarizing the detection capabilities of each technique against different attack types. The symbols √, ×, and □ are used to indicate whether a technique can successfully detect all attacks of a particular type, cannot detect all attacks, or can only partially detect due to limitations.

A similar analysis is done for SQL injection prevention techniques, w showcasing the prevention capabilities of each technique against different attack types. Again, symbols are used to indicate the prevention effectiveness.

The addressing percentage of SQL injection attacks among detection and prevention techniques is also calculated based on the techniques' ability to detect or prevent specific attack types.

Additionally, the review provides a comparison of the detection and prevention techniques based on deployment and evaluation criteria, such as detection time, detection location, code modification requirements, and additional infrastructure needed.

In conclusion, the literature review presents an assessment of SQL injection attacks and various detection and prevention techniques. It identifies different types of attacks, explores the techniques available, and compares them based on their capabilities and deployment requirements.

Research Paper 7:-

Title : SQL Injection Attack Scanner Using Boyer-Moore String Matching Algorithm.

Author : Teh Faradilla Abdul Rahman, Alya Geogiana Buja, Kamarularifin Abd. Jalil, Fakariah Mohd Ali

Inference:

An excerpt from the research paper on web vulnerability scanning and the use of the Boyer-Moore String Matching algorithm to detect SQL injection attacks, over here the introduction highlights the risks associated with web applications and the prevalence of SQL injection as a method for cyber-attacks. It emphasizes the limitations of existing web vulnerability scanners and the need for a more efficient and accurate approach.

The section on SQL injection explains the nature of this vulnerability and its potential impact on databases. It emphasizes the importance of input validation to prevent SQL injection attacks. The methodology section presents examples of vulnerable and non-vulnerable web pages and discusses different exact string matching algorithms, including Brute-Force, Knuth-Morris-Pratt, and Boyer-Moore.

The Boyer-Moore algorithm is chosen for its efficiency in searching for patterns and its suitability for scanning web applications. The proposed model based on this algorithm is described, including the flow of the scanning process through various panels, such as the crawler, parameter testing, exploit, and report panels.

The findings and discussions section introduces the SDR web vulnerability scanner, which was developed based on the proposed model. It provides a demonstration of the scanner's functionality and shows examples of the output from different panels. The efficiency and accuracy of the model are evaluated, considering the time taken for crawling and parameter testing. Results are presented in a table and a graph.

In conclusion, the proposed model using the Boyer-Moore algorithm is effective in detecting vulnerable web applications susceptible to SQL injection attacks. The implementation of the model can assist web developers and administrators in securing their applications. Further improvements can be made by adding additional parameters to enhance the accuracy of the model. Overall, the research contributes to the field of web vulnerability scanning and highlights the importance of addressing SQL injection vulnerabilities.

Research Paper 8:-

Title : Preventing SQL injection attack using pattern matching algorithm.

Author : Swapnil Kharche, Jagdish patil Kanchan Gohad, Bharti Ambetkar

Inference:

The literature review discusses the challenges faced by internet applications that utilize databases in terms of security and protection of private data, particularly focusing on the threat of SQL Injection attacks. It emphasizes that SQL Injection attacks pose a serious risk to web application security by targeting databases accessed through web front-ends and exploiting flaws in input validation logic.

The review highlights that current web applications and detection systems lack comprehensive knowledge of attacks and rely on limited sets of attack patterns for evaluation. It points out that SQL Injection attacks have resulted in serious consequences, such as unauthorized access to sensitive information, fraudulent activities, and data leaks.

Previous research in the field is briefly mentioned, including a framework for detecting malicious database transaction patterns using data mining, an enhanced model for identifying intruders in databases without role-based access control, and techniques based on anomaly detection and association rule mining to identify abnormal query behaviour.

The proposed scheme for detecting and preventing SQL Injection attacks is introduced, which utilizes both a static phase and a dynamic phase. In the static phase, a static pattern matching algorithm based on the Aho-Corasick algorithm is applied to identify known anomaly patterns. The detected anomalies are maintained in a static pattern list. In the dynamic phase, if a new anomaly occurs, an alarm is triggered, and a new anomaly pattern is generated and added to the static pattern list.

The calculation of an anomaly score value is explained, where the score is based on the similarity between the user-generated SQL query and the patterns in the static pattern list. If the score exceeds a threshold value, the query is forwarded to the administrator for further analysis.

The Aho-Corasick algorithm is described as a method for recognizing patterns using finite automata. It constructs a finite automaton during pre-computation using the set of anomaly keywords, allowing for efficient pattern matching of SQL queries.

The proposed architecture and algorithm are illustrated through figures depicting the SQL query generation process, the pattern matching process, and examples of exact pattern matching.

In conclusion, the paper emphasizes the importance of protecting internet applications that use databases from SQL Injection attacks, which can compromise data confidentiality and have severe financial and reputational consequences. The proposed scheme using Aho-Corasick pattern matching algorithm shows promise in detecting and preventing SQL Injection attacks, with initial evaluations indicating no false positives or false negatives.

Research Paper 9:-

Title : A Systematic Literature Review on SQL Injection Attacks Techniques and Common Exploited Vulnerabilities

Author : Salem A. Faker , Mohamed A. Muslim and Harry S. Dachlan

Inference:-

It can be inferred that this study conducted a systematic literature review (SLR) on SQL injection attacks (SQLIA) to understand their impacts, types, and mechanisms. The SLR consisted of planning, conducting, reporting, and discussing phases. The researchers formulated research questions and performed an advanced search in prominent digital libraries, covering articles published from 2010 to 2017. They applied inclusion and exclusion criteria to select 46 primary studies for analysis. SQLIA involves exploiting vulnerabilities in user input handling to execute unauthorized SQL queries, leading to data loss, authentication bypass, and database destruction.

SQLIA can take different forms, including identifying injectable parameters, extracting/modifying data, denial of service, authentication bypass, remote command execution, and privilege escalation. Attackers employ mechanisms like tautologies, piggy-backed queries, alternate encodings, illegal/logical queries, union queries, stored procedures, and inference-based attacks. The discussion highlights the real-world impacts of SQLIA and emphasizes the need for robust security measures to prevent and mitigate such attacks. The study suggests implementing input filtering, parameterized queries, and secure coding practices to mitigate the risks associated with SQLIA and protect databases from unauthorized access and manipulation.

Research Paper 10:-

Title : SQL INJECTION DETECTION USING HYBRID MODEL

Author : Dr. Sandeep Kumar , Tanya Ahuja, Bhavya Choudhary

Inference:-

The paper focuses on detecting SQL injection attacks using basic Machine Learning algorithms, specifically Logistic Regression, KNN, and LDA. Stacking technique is employed to improve accuracy, with Logistic Regression as the meta model. The dataset consists of labeled SQL queries, and performance metrics like accuracy are evaluated. Logistic Regression and Neural Networks perform best. The paper concludes that SQL injection attacks pose security risks, and detection algorithms help mitigate them. Advantages include not needing large datasets or high computational power. Stacking combines predictions for a more robust system. Future work includes extending to other web attacks and improving accuracy. Dataset is from Kaggle. Usability and efficiency can be enhanced with better feature selection and training approaches. Validation techniques like cross-validation are suggested.

Research Paper 11:-

Title :
A Literature Review and Comparative Analyses on SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques

Author : Bojken Shehu and Aleksander Xhuvani

Inference:

SQL injection is a severe security vulnerability that targets the database layer of an application by exploiting weak input validation. It allows attackers to gain unauthorized access to the application's database and manipulate the SQL queries generated by the application. Over the years, various types of SQL injection attacks have emerged, making it crucial to understand and identify these techniques to effectively counter them.

Common types of SQL injection attacks include tautologies, where the attacker injects code in conditional statements to always evaluate as true, bypassing authentication or extracting data. Logically incorrect query attacks aim to gather information about the database by generating error messages with useful debugging information. Union queries trick the database into returning results from unintended tables by joining injected queries with safe ones using the UNION keyword.

Stored procedures attacks exploit vulnerabilities in database-stored procedures to perform privilege escalation and execute malicious SQL procedures. Piggy-backed queries involve injecting additional queries alongside the original one, causing the database to execute multiple SQL queries. Inference attacks, such as blind injection and timing attacks, exploit the database's response time or error messages to extract information. Alternate encodings involve modifying injection queries using alternate encoding formats to evade detection.

Addressing SQL injection requires a combination of preventive and detection techniques. Input validation and sanitization of user inputs, utilizing parameterized queries or prepared statements, and implementing strict access controls are important preventive measures. Intrusion detection systems (IDS) and web application firewalls (WAF) can help detect and block SQL injection attempts. However, it is essential to stay updated on the latest attack techniques and employ comprehensive security measures to mitigate the risk of SQL injection effectively.

Research Paper 12:-

Title : Generation of SQL-injection free secure algorithm to detect and prevent SQL-injection attacks

Author : Kanchana Natarajana , Sarala Subramani

Inference:-

This inference paper discusses the development of a secure algorithm to detect and prevent SQL-injection attacks in web applications. It highlights the significant risks posed by SQL-injection attacks to commercial vendors and emphasizes the need for robust security measures. The limitations of existing vulnerability detection scanners are acknowledged, and the challenges faced by security-oriented developers in creating reliable tools are mentioned.

Several methods for detecting and preventing SQL-injection attacks are explored, including static analysis, dynamic analysis, and combined approaches. The advantages of dynamic analysis tools, such as penetration testing, are emphasized for their ability to identify vulnerabilities during runtime. The AMNESIA tool, which combines static and dynamic analysis, is presented as an effective method for building a model of legitimate queries and detecting deviations at runtime.

The excerpt briefly mentions the limitations of web framework methods in filtering user input parameters and the drawbacks of machine learning techniques, including high false positives and low detection rates. The importance of proper input sanitization, syntax validation, and adherence to security guidelines during the programming phase is emphasized for mitigating SQL-injection vulnerabilities.

Research Paper 13: -

Title: Review of SQL Injection Attack and Proposed Method for Detection and Prevention of SQLIA

Author: Mayank Namdev, Fehreen Hasan, Gaurav Srivastav

Inference: -

Among several researches based on the SQL injection Prevention & Attacks, it was found that certain cases are there, where these approaches are not found to be effective. Hence, these approaches become un-useful cannot able to detect the injections to prevent them. In addition, the attackers can access the database directly in an illegal way. Therefore, here proposed a new approach that is completely based on the hash method of using the SQL queries in the web-based environment, which is much secure and provide the prevention from the attackers SQL. But this proposed strategy requires the alterations in the design of existing schema database and a new guideline for the database user before writing any new database. Through these guidelines, we expect the effective outcomes in SQL injections Preventions. In addition, many researchers can contribute to this SQL Injections Attacks Preventions Mechanisms without any extra effort to the database like alteration in schema design, adding new tables, adding new attributes etc.

Research Paper 14: -

Title: AN EFFICIENT TECHNIQUE FOR SQL INJECTION DETECTION AND PREVENTION

Author: Esraa Mohamed Safwat, Hany Mahgoub, ASHRAF EL-SISI, Arabi Keshk

Inference: -

New server-side protection technique against SQL Injection, which is designed to check for SQL injection vulnerabilities in the server side. SQLI-DP intercepts SQL queries and analyze it based on the syntax of the SQL queries using parse tree. It rejects injectable queries from beginning and executes other queries to detect SQL injection. It is suitable for legacy system because it is a technique that is implemented on server side as it detects blind SQL injection using new function. Using parse tree to detect SQL injection using Zql [25] with open-source technique. Detect and prevent the SQL injection using this leads to decrease execution time. This technique has two options (continue with safety, unsafe option) if we select the safety option activate SQLI-DP. If no (unsafe option) send request immediately to database and execute query. Illustrates the core work of SQLI-DP technique where the three points.

SQLI-DP has two advantages comparing with another scanner. It is efficient, adding about 1second overhead to database query costs. In addition, it is easily adopted by software developer, having the same syntactic structure as current popular record set retrieval method and we detect and prevent blind SQL injection

----

Research Paper 15: -

Title: SQL Injection Prevention Using Random4 Algorithm

Author: Rakhangi Aasim Hanif Munira, Shaikh Abedulla Dastageer Nasrin

Inference:

The random4 algorithm is based on randomization and is used to convert the input into a cipher text incorporating the concept of cryptographic salt. This algorithm forms the basis of the proposed approach. Any input in web forms will contain numbers, uppercase, lowercase, or special characters. Keeping this in mind the input from user is encrypted based on randomization. In our algorithm the valid inputs are numbers, lowercase, or uppercase characters and at most 10 special characters. The reason for choosing only 10 special characters is that they are rarely used and for additional security. Each character in the input can have 72 combinations (26 lowercase, 26 uppercase, 0-9 and 10 special characters). Hence for a 6-character input there can be 726 combinations possible. To encrypt the input, each input character is given four random values.

P.no	Title	Author	Algorithm	Advantages	Disadvantages	Tool

1	**SQL Injection Prevention by Adaptive Algorithm**	**Ashish John - Dept. of Computer Science and Engineering, SRM University, NCR Campus**	Adaptive Algorithm	Simplicity and efficiency, making it suitable for large datasets	Results can change depending on where we start or place the initial groups.	Python
2	**Prevention Of Sql Injection Attack Using Unsupervised Machine Learning Approach**	**M.N.Kavitha, V. Vennila, G. Padmapriya, A. Rajiv Kannan**	Random Forest algorithm	Ability to handle high-dimensional data and maintain good predictive performance even when there are irrelevant features	Can be computationally expensive	Scikit-learn

3	**SQL Injection Attacks Predictive Analytics Using Supervised Machine Learning Techniques**	**Akinsola, Jide E. T, Awodele, Oludele, Idowu, Sunday, Kuyoro, Shade**	Naive Bayes algorithm	Works well with high-dimensional datasets	It assumes independence between features, which may not always hold true.	Scikit-learn
4	**Review of SQL Injection Attack and Proposed Method for Detection and Prevention of SQLIA**	**Mayank Namdev ; Fehreen Hasan ; Gaurav Shrivastav**	Long Short-Term Memory	Capable of learning long-term dependencies in sequential data	Computationally expensive and time-consuming	PyTorch

5	**A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm.**	**Oluwakemi Christiana Abikoye , Abdullahi Abubakar , Ahmed Haruna Dokoro , Oluwatobi Noah Akande and Aderonke Anthonia Kayode.**	**Knuth Morris Pratt string match algorithm.**	1)Efficiency: The KMP algorithm is known for its efficient string matching capability. 2)Accuracy	1) Limited scope: While the KMP algorithm can efficiently detect specific patterns. 2)False positives	Programming Language,Text Processing Library or Functions,Security Framework,Database Access Layer,Web Application Firewall
6	**A Literature Review and Comparative Analyses on SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques**	**Bojken Shehu and Aleksander Xhuvani**				WAVES and WebSSARI

7	**SQL Injection Attack Scanner Using Boyer-Moore String Matching Algorithm.**	**Teh Faradilla Abdul Rahman, Alya Geogiana Buja, Kamarularifin Abd. Jalil, Fakariah Mohd Ali**	**Boyer-Moore String Matching Algorithm.**	1)Efficiency: It can quickly identify and locate specific patterns in a given text by utilizing precomputed tables and smart character comparisons. 2) High Accuracy: The Boyer-Moore algorithm has a high probability of accurate pattern matching.	1)Limited Detection Scope: It relies on predefined patterns to search for exact matches, which means it may struggle with detecting attacks that utilize variations, obfuscation techniques, or non-standard patterns. 2) False Negatives for Encrypted or Encoded Data	Programming language, text processing library or Functions, SQL Injection Attack Patterns, Database Connectivity, Input Data Source, Reporting or Alerting Mechanism.
8	**Preventing SQL injection attack using pattern matching algorithm.**	**Swapnil Kharche , Jagdish patil Kanchan Gohad, Bharti Ambetkar**	**Pattern matching algorithm**	1)Accuracy in Detection: The algorithm searches for predefined patterns or signatures commonly associated with SQL injection attacks, allowing for precise identification of	1)Limited Coverage of Unknown or Evolving Attack Patterns: While pattern matching algorithms excel at detecting known attack patterns, they may struggle with	Programming language, text processing library or Functions, SQL Injection Attack Patterns, Database Connectivity, Input Data Source,

				malicious input. 2) Customizability and Flexibility: A pattern matching algorithm for SQL injection prevention can be tailored to fit the specific requirements and context of an application	detecting new or evolving attack techniques that don't match the predefined patterns. 2) False Negatives and False Positives	Reporting or Alerting Mechanism.
9	**A Systematic Literature Review on SQL Injection Attacks Techniques and Common Exploited Vulnerabilities**	**Salem A. Faker , Mohamed A. Muslim and Harry S. Dachlan**				

10	**SQL INJECTION DETECTION USING HYBRID MODEL**	**Dr. Sandeep Kumar , Tanya Ahuja, Bhavya Choudhary**	Logistic Regression, K-Nearest Neighbor (KNN), Linear Discriminant Analysis (LDA), Stacking.	Using basic Machine Learning algorithms allows for efficient detection of SQL injection attacks.	1) The performance of the detection system heavily relies on the quality And representativeness of the dataset used. 2)Limited to using Machine Learning algorithms, which may have limitations in handling complex and evolving attack patterns.	mysql
11	**A Literature Review and Comparative Analyses on SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques**	**Bojken Shehu and Aleksander Xhuvani**	SQL-Injection Free (SQL-IF) Secure Algorithm	1)Comprehensive coverage 2) Identification of vulnerabilities	Performance impact and Complexity	SQLMap, Web Application access control

12	**Generation of SQL-injection free secure algorithm to detect and prevent SQL-injection attacks**	**Kanchana Natarajana , Sarala Subramani**	SQL-Injection Free (SQL-IF) Secure Algorithm	1)Increased security 2)unauthorized access and data breaches 3)Improved data integrity 4)Flexibility and adaptability	Performance impact and Complexity	Database security tools, Code analysis tools
13	**Review of SQL Injection Attack and Proposed Method for Detection and Prevention of SQLIA**	**Mayank Namdev, Fehreen Hasan, Gaurav Srivastav**	HASH FUNCTION ALGORITHM	Data integrity checks, Password storage, Detection of data tampering, Data validation and consistency	Irreversibility, Hash collisions, Performance impact, Storage requirements	SQL-MAP

14	**AN EFFICIENT TECHNIQUE FOR SQL INJECTION DETECTION AND PREVENTION**	**Esraa Mohamed Safwat, Hany Mahgoub, ASHRAF EL-SISI, Arabi Keshk**	SQLI-DP	No need to rewrite old web application, it's efficient, same syntactic structure	False Positives/Negatives, Complexity and Overhead, Limited Coverage	MYSQL
15	SQL Injection Prevention Using Random4 Algorithm	Rakhangi Aasim Hanif Munira, Shaikh Abedulla Dastageer Nasrin	Random4 algorithm	Randomization, Cryptographic Salt, Wide Character Support, Increased Combinations	Custom Algorithm, Security Evaluation, Limited Special Character Support, Key Management, Performance Overhead	Apache Tomcat 7, Eclipse

Detection

Methodology 1:-

Review of SQL Injection Attack and Proposed Method for Detection and Prevention of SQLIA

Methodologies:

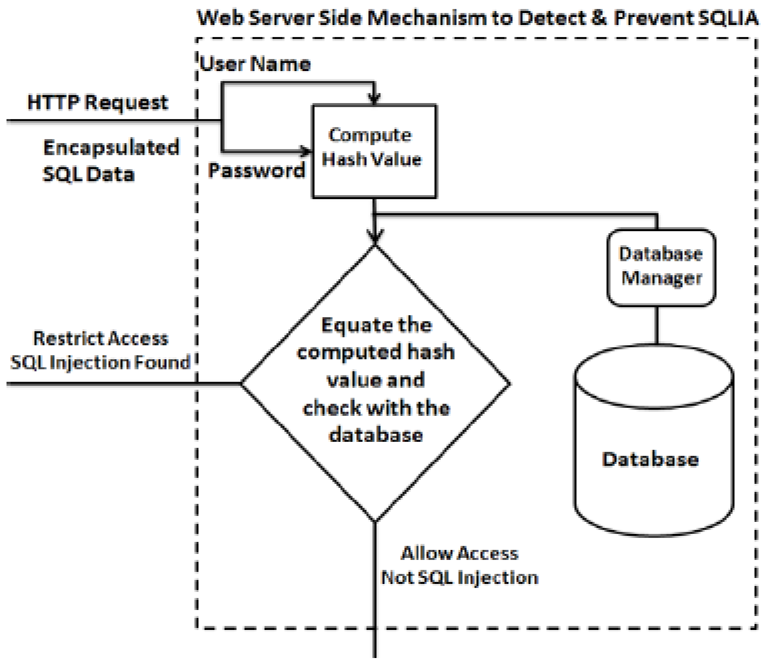

Web Server Side Mechanism to Detect & Prevent SQLIA
User Name
HTTP Request
Encapsulated SQL Data
Password
Compute Hash Value
Database Manager
Restrict Access SQL Injection Found
Equate the computed hash value and check with the database
Database
Allow Access Not SQL Injection

Figure 3 : Proposed Hash Scheme for Detecting SQLIA & Prevent Them

SQL Injection Protector for Authentication (SQLIPA) uses for preventing database against SQL injection. In the proposed approach there is a need for two extra columns in database. The first one is for the hash values of user name and other one for the hash values of password. The hash values are calculated for user name and password when a user account is created for the first time and stores it in the User table. Whenever user wants to login to database his/her identity is checked using user name and password and its hash values. These hash values are calculated at runtime using store procedure when user wants to login into the database.

In the proposed technique first the hash values of user name and password are calculated at runtime and checked with stored hash values in the database table. During the authentication the original query will change into a new modified query with hash parameters.

Hence, if a user tries the injection to the query, and our proposed methodology is working with SQL query, it will automatically detect the injections as the potentially harmful content and rejects the values. Therefore, it cannot bypass the authentication process. The advantage of the proposed technique is that the hackers do not know about the hash values of user name and password. So, it is not possible for the hacker to bypass the authentication process through the general SQL injection techniques. The SQL injection attacks can only be done on codes which are entered through user entry form but the hash values are calculated at run time at backend before creating SELECT query to the underlying database therefore the hacker cannot calculate the hash values as it dynamic at Runtime.

Hence to prevent after-login attacks we have taken the help of data encryption. As we saw in the previous section that it is possible to collect the highly confidential information by using **union** operator we find out an alternative way to store all these confidential information"s. In our database, instead of directly storing all confidential information"s, we store them in encrypted format with a secure and confidential encryption-key. Now even if the dispatcher user can able to see the atm_pin by using union operation, he cannot able to decrypt it without knowing the exact encryption method and encryption- key. So he cannot able to do any damage with that encrypted atm_pin.

Implementation:

We try to illustrate our technique using an example. Suppose the given query is:

SELECT username FROM accounts WHERE

username=" or 1=1 #' AND password=".

This is a typical SQLIA, whose '#' symbol comments out the query behind it, the capital words are terminal nodes recognized by the lexer. The non-capital words are syntax rules' names recognized by the parser. Notice that the comment does not show in the AST, this is because the lexer in Antlr put all the non-related symbols in a hidden-channel; in other words, the lexer filters out non-related symbols, such as comment. That is why our proposed method could still work for SQLIAs with bypassing techniques.

Word embedding: Given the AST, we then used the DFS algorithm to traverse the tree, thus converting the AST to a symbol sequence. Using DFS, we could preserve the context information of the syntax tree., this would be ['dmlStatement', 'selectStatement', 'querySpecification', 'SELECT', …, 'decimalLiteral', 'DECIMAL_LITERAL'].

Suppose an SQL query set $Q = \{q_1, q_2, \ldots, q_n\}$ has n SQL queries, and it contains K unique symbols. We utilized a continuous bag of words (CBOW) as the word embedding method to learn the word embedding vector for each unique symbol. The reason why we chose CBOW is that we think the symbol in the sequence could be predicted with probabilities using other symbols around it., each symbol in the AST is transformed to word vector, by stacking them together we have the word embedding matrix.

Building LSTM network: The detailed hyperparameters for our designed model are listed in Table 1. We assume that the maximum length of the symbol sequence is L in our training set. For batch training convenience, for each SQL query, we padded it to the length L and fed it into the LSTM network. Note that we added a dropout layer in the network, and we also trained our LSTM network with two other dropout hyperparameters, namely d1 for LSTM input gate dropout and d2 for recurrent connection dropout. These setups would help us to prevent over-fitting problems.

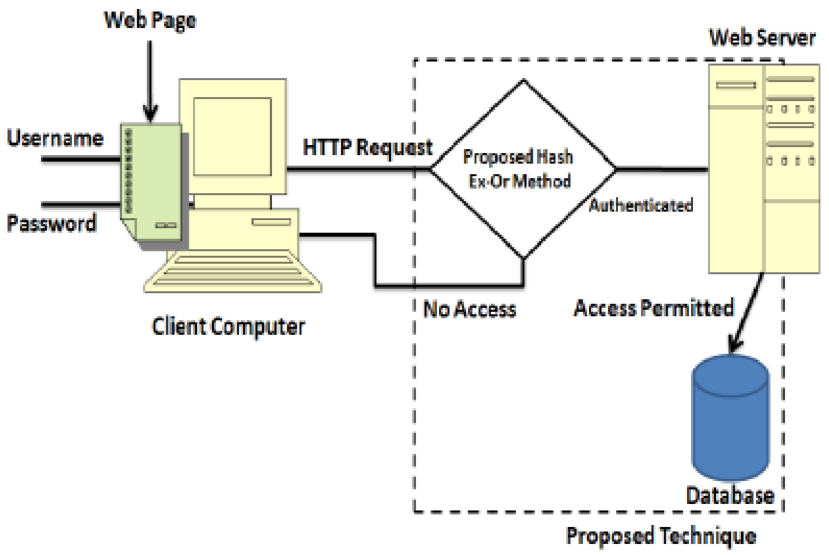


Figure 2 : System design for SQLIA Detection & Prevention

Methodology 2:-

Generation of SQL-injection free secure algorithm to detect and prevent SQL-injection attacks:

METHODOLOGY:

The below shown Algorithm1 performs its function by assigning the Form Status *(FS)* as attack and free with collection of fields obtained from the collection of forms. The Form *(fm')* is obtained from the set of forms *(Fm)* whereas values of every field are obtained from the form *(Fm')*. Three well-defined

functions are generated inside the method called CheckVulnerability (f') to check for any special

characters, keywords and Boolean characters.

Algorithm 1: Proposed SQL-IF Secure Algorithm

Algorithm SQLIAD (Form F)

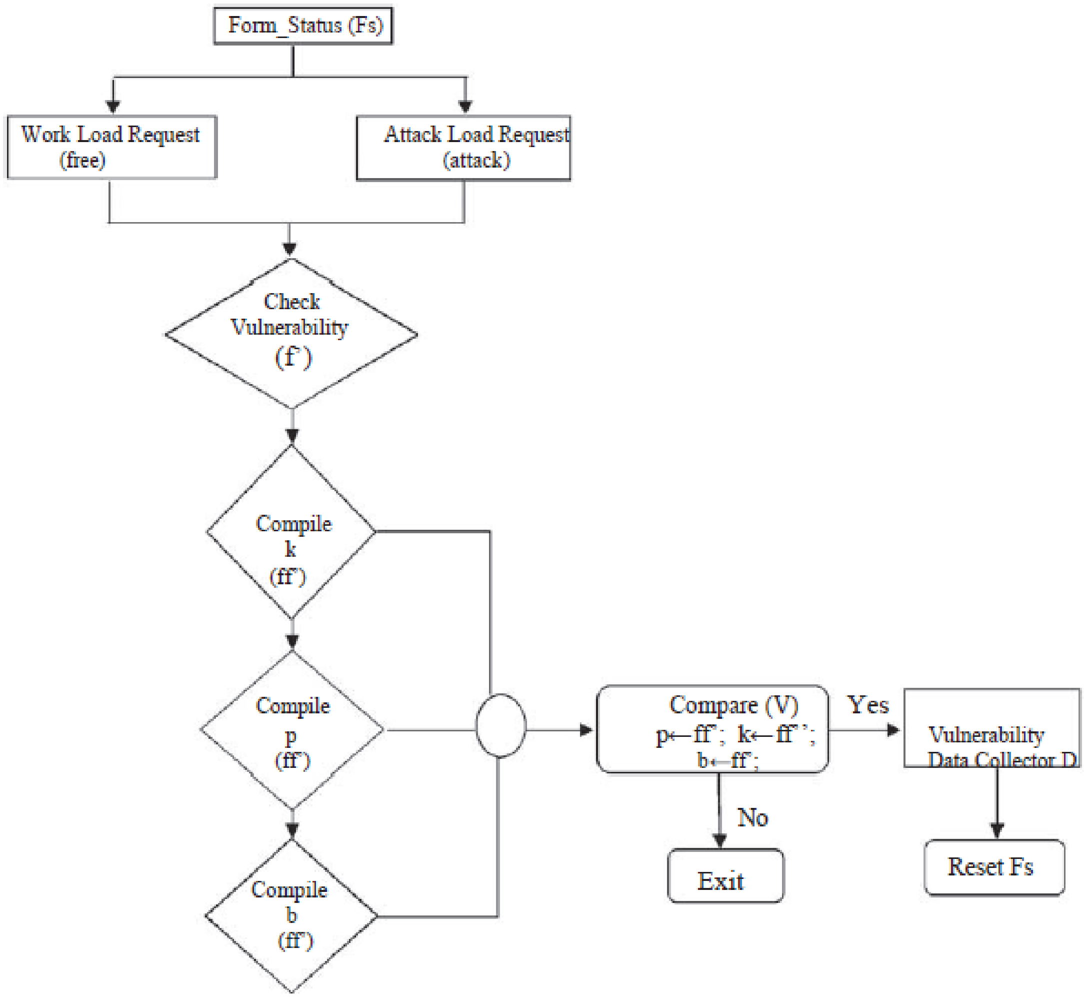
01 Input: Fm denotes the collection of forms with collection of Fields.

```
02 Input: Enumerate Form_Status FS = {attack, free}
03 Input: Default Value to FS as free
04 Output: FS
05 for each fm' □ Fm do
06 for each f’ □ fm’ do
07 f’ □ fields contains the values
08 if f’ is not empty string then
09 FS □ output from the method CheckVulnerability (f’)
10 if FS as attack
11 D □ Add the field f’ in the collection
12 Reset the Http requests to issue warning;
13 return FS;
```

Generic detection method 1: Generated to check for Special characters, keywords and Boolean keywords

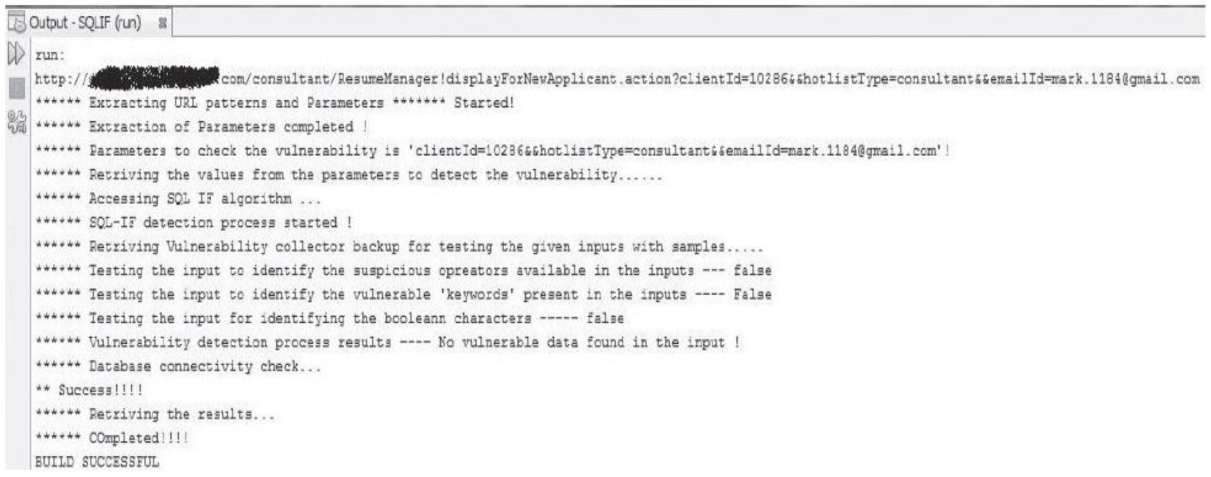
```
CheckVulnerability (f’):
for each f’ fields do
if f’ is a non-empty string then
// to check for special characters in the input fields and parameters
p = collection of compiled special characters like {(([',&+=<><=>=])}
for each tokens ff’ □ f’ then
ff” = compile ff’ to make all the input tokens neutralize.
v = comparison of P and ff’;
if v is not true then return v;
// to check for keywords in the input fields and parameters
```

```
k = collection of keywords
{union|select|intersect|insert|update|delete|drop|truncate}
for each tokens ff’ □ f’ then
ff” = compile ff’ to make all the input tokens neutralize.
v = compare ff” with k;
if v is not true then **return v**;
// to check for Boolean characters in the input fields and parameters
b = collection of Boolean characters {' or '|'or'|'AND'|'and'"}
for each tokens ff’ □ f’ then
ff” = compile ff’ to make all the input tokens neutralize.
v = compare b with ff’;
return v;
```

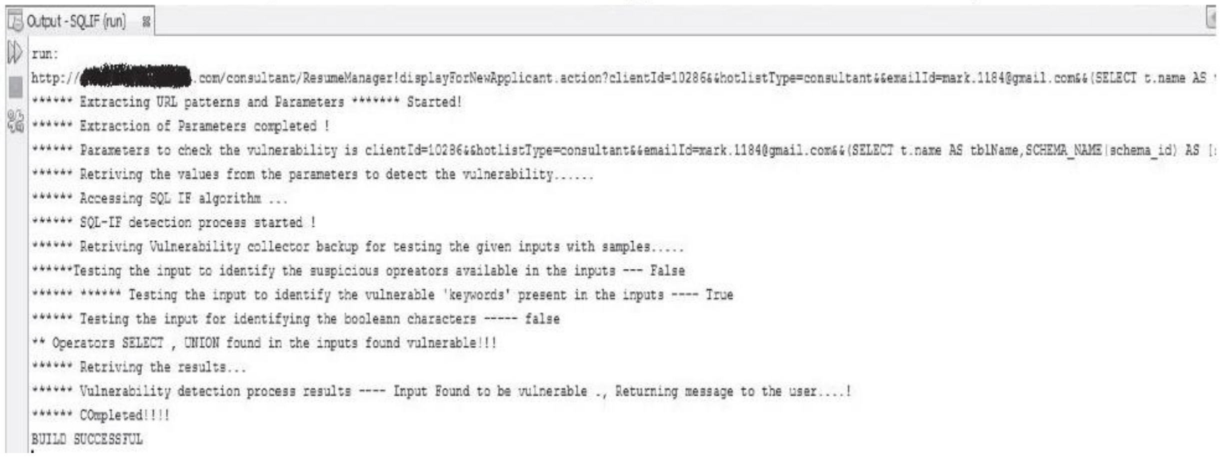


Implementation:

The implementation of the proposed SQL-injection free technique has been done under Java Programming Language and Structured Query Language. The proposed technique is generic in any web applications and it is language independent, can be implemented in any programming language. Source code analysis plays a major role in the detection of SQL injection attacks. At the initial stage, we implemented the algorithm in the real time web application and the results obtained are positive. Since the web application is free from SQLIAs and further analyses with vulnerable web application is required.


Output - SQLIF (run)

```
run:
http://[redacted]com/consultant/ResumeManager!displayForNewApplicant.action?clientId=10286&&hotlistType=consultant&&emailId=mark.1184@gmail.com
****** Extracting URL patterns and Parameters ******* Started!
****** Extraction of Parameters completed !
****** Parameters to check the vulnerability is 'clientId=10286&&hotlistType=consultant&&emailId=mark.1184@gmail.com'!
****** Retriving the values from the parameters to detect the vulnerability......
****** Accessing SQL IF algorithm ...
****** SQL-IF detection process started !
****** Retriving Vulnerability collector backup for testing the given inputs with samples.....
****** Testing the input to identify the suspicious opreators available in the inputs --- false
****** Testing the input to identify the vulnerable 'keywords' present in the inputs ---- False
****** Testing the input for identifying the booleann characters ----- false
****** Vulnerability detection process results ---- No vulnerable data found in the input !
****** Database connectivity check...
** Success!!!!
****** Retriving the results...
****** COmpleted!!!!
BUILD SUCCESSFUL
```


Output - SQLIF (run)

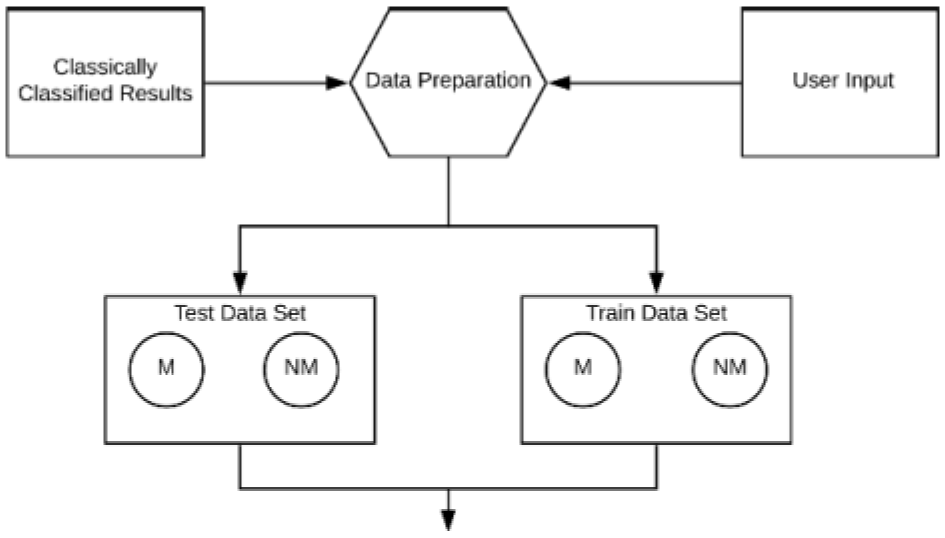
```
run:
http://[redacted].com/consultant/ResumeManager!displayForNewApplicant.action?clientId=10286&&hotlistType=consultant&&emailId=mark.1184@gmail.com&&(SELECT t.name AS
****** Extracting URL patterns and Parameters ******* Started!
****** Extraction of Parameters completed !
****** Parameters to check the vulnerability is clientId=10286&&hotlistType=consultant&&emailId=mark.1184@gmail.com&&(SELECT t.name AS tblName,SCHEMA_NAME(schema_id) AS [
****** Retriving the values from the parameters to detect the vulnerability......
****** Accessing SQL IF algorithm ...
****** SQL-IF detection process started !
****** Retriving Vulnerability collector backup for testing the given inputs with samples.....
******Testing the input to identify the suspicious opreators available in the inputs --- False
****** ****** Testing the input to identify the vulnerable 'keywords' present in the inputs ---- True
****** Testing the input for identifying the booleann characters ----- false
** Operators SELECT , UNION found in the inputs found vulnerable!!!
****** Retriving the results...
****** Vulnerability detection process results ---- Input Found to be vulnerable ., Returning message to the user....!
****** COmpleted!!!!
BUILD SUCCESSFUL
```

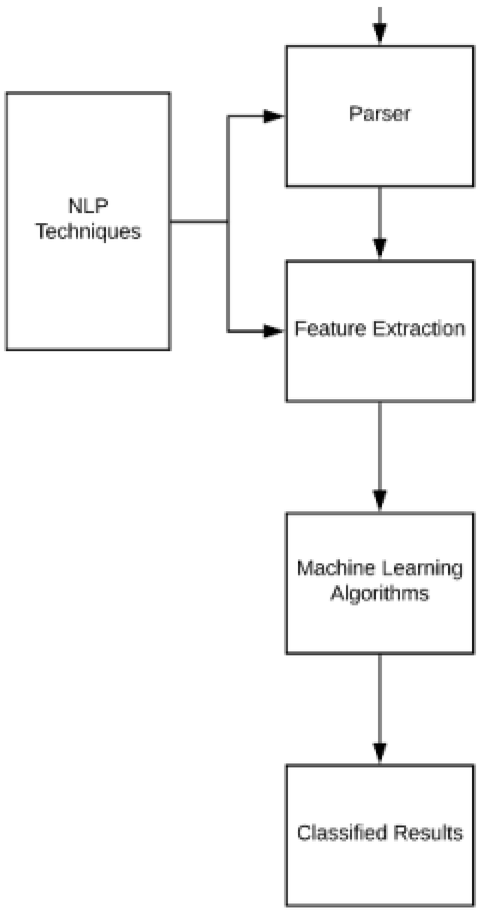
At the next stage the proposed generic code has been tested with vulnerable web application and the results obtained are negative. Hence the proposed algorithm filtered the Union Queries and some special keywords which are found to be vulnerable input to the database.

Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review

Methodology 3:-

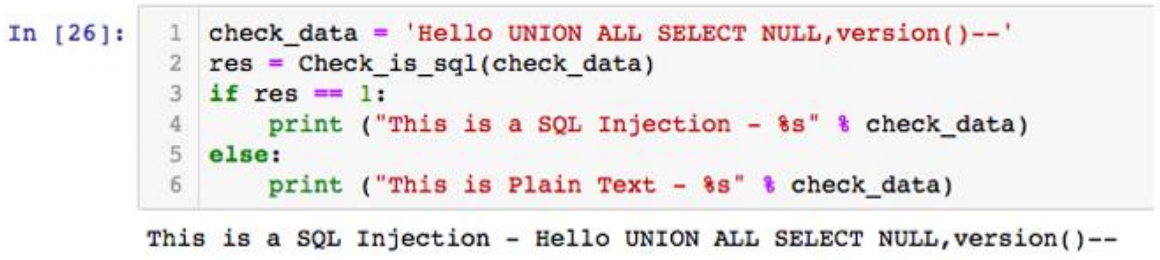
Gradient boosting is an Ensemble learning method to reduce errors and provide predictions with better accuracy. Gradient Boosting algorithm uses simple classifiers, mostly decision trees, in a sequential manner, to provide results. The algorithm first uses the simple classifier to classify the data. Then the results are considered to calculate the errors or the data points that were not easily fit by the simple classifier. The algorithm then focuses on those data points in the next round and tries to fit them as well. In this way the errors are reduced, and outlier data points are also taken into consideration. But overdoing this remodeling can also cause overfitting. Hence learning to stop remodeling at an acceptable accuracy and error rate is also an important point to consider with this approach. In this paper, we use Gradient Boosting machine learning approach to detect and prevent SQL Injections.




NLP Techniques
Parser
Feature Extraction
Machine Learning Algorithms
Classified Results

Implementation:

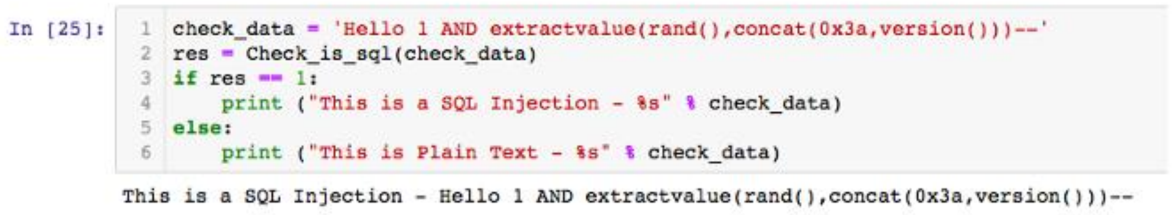
Identifying Union Based SQL Injection:



```
In [26]:
check_data = 'Hello UNION ALL SELECT NULL,version()--'
res = Check_is_sql(check_data)
if res == 1:
    print ("This is a SQL Injection - %s" % check_data)
else:
    print ("This is Plain Text - %s" % check_data)
```

```
This is a SQL Injection - Hello UNION ALL SELECT NULL,version()--
```

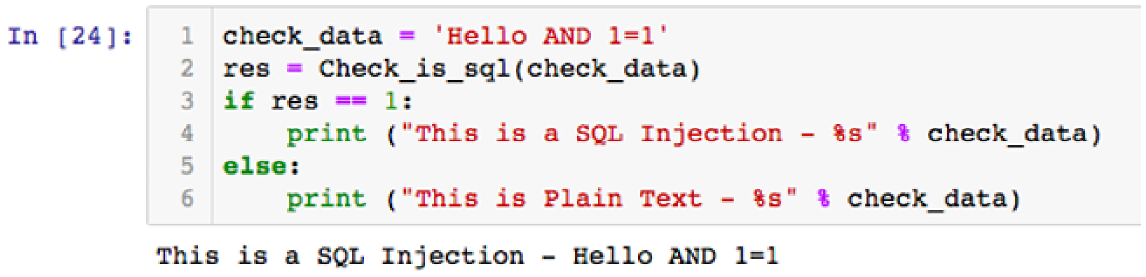
Identifying Error Based SQL Injection:



```
In [25]:
check_data = 'Hello 1 AND extractvalue(rand(),concat(0x3a,version()))--'
res = Check_is_sql(check_data)
if res == 1:
    print ("This is a SQL Injection - %s" % check_data)
else:
    print ("This is Plain Text - %s" % check_data)
```

```
This is a SQL Injection - Hello 1 AND extractvalue(rand(),concat(0x3a,version()))--
```

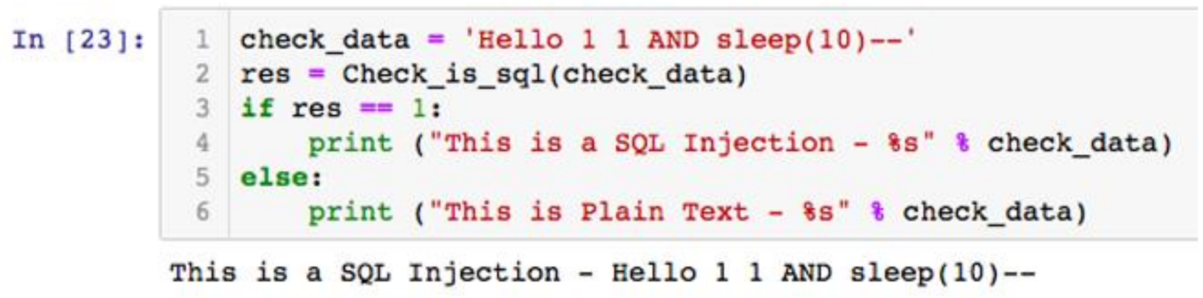
Identifying Boolean SQL Injection:



```
In [24]:
check_data = 'Hello AND 1=1'
res = Check_is_sql(check_data)
if res == 1:
    print ("This is a SQL Injection - %s" % check_data)
else:
    print ("This is Plain Text - %s" % check_data)
```

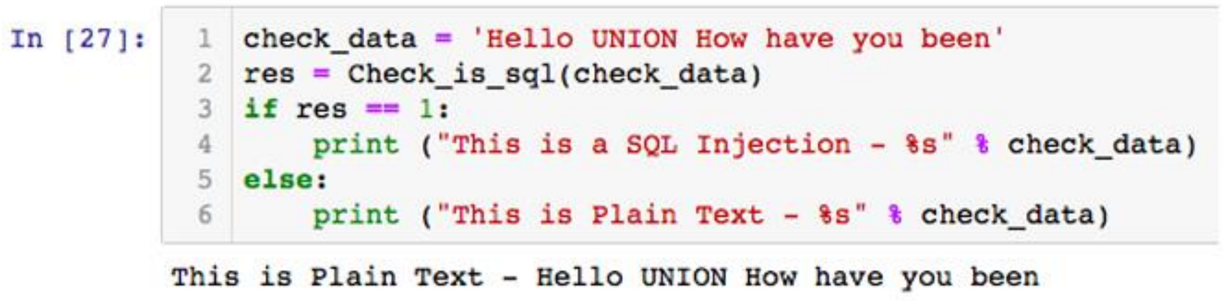
```
This is a SQL Injection - Hello AND 1=1
```

Identifying Time Based SQL Injection:



```
In [23]:
check_data = 'Hello 1 1 AND sleep(10)--'
res = Check_is_sql(check_data)
if res == 1:
    print ("This is a SQL Injection - %s" % check_data)
else:
    print ("This is Plain Text - %s" % check_data)

This is a SQL Injection - Hello 1 1 AND sleep(10)--
```

Identifying Plain-text:



```
In [27]:
check_data = 'Hello UNION How have you been'
res = Check_is_sql(check_data)
if res == 1:
    print ("This is a SQL Injection - %s" % check_data)
else:
    print ("This is Plain Text - %s" % check_data)

This is Plain Text - Hello UNION How have you been
```

A Literature Review and Comparative Analyses on SQL Injection: Vulnerabilities, Attacks and their Prevention and Detection Techniques

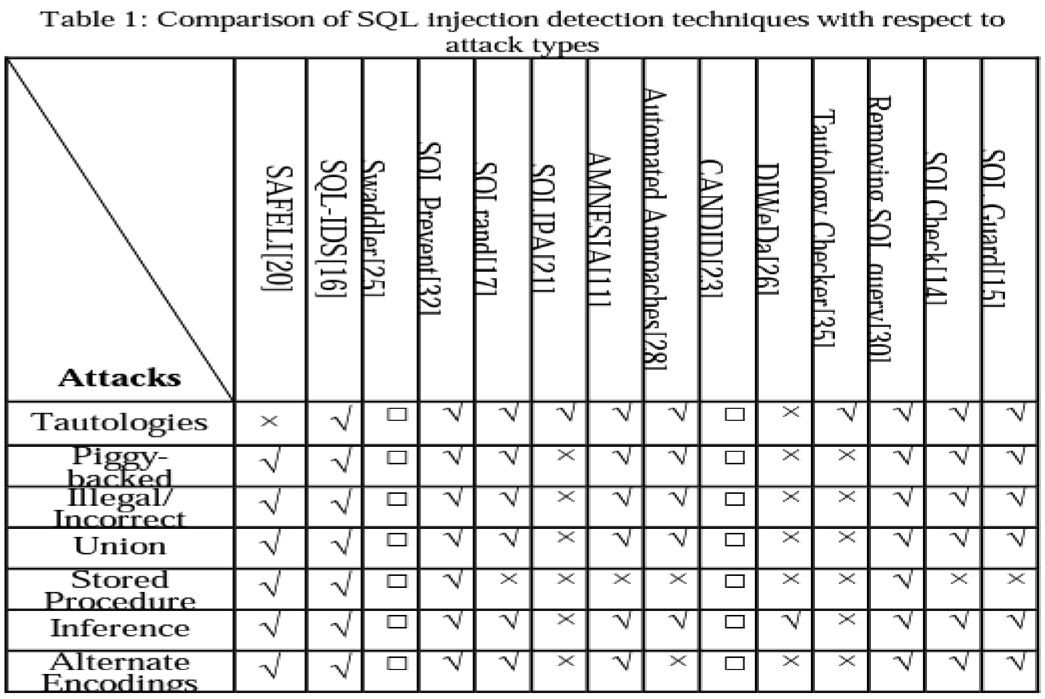


Table 1: Comparison of SQL injection detection techniques with respect to attack types

Attacks	SAFELI[20]	SQL-IDS[16]	Swaddler[25]	SQL Prevent[32]	SQLrand[17]	SQLIPA[21]	AMNESIA[11]	Automated Approaches[28]	CANDID[23]	DIWeDa[26]	Tautology Checker[35]	Removing SQL query[30]	SQL Check[14]	SQL Guard[15]
Tautologies	×	√	□	√	√	√	√	√	□	×	√	√	√	√
Piggy-backed	√	√	□	√	√	×	√	√	□	×	×	√	√	√
Illegal/ Incorrect	√	√	□	√	√	×	√	√	□	×	×	√	√	√
Union	√	√	□	√	√	×	√	√	□	×	×	√	√	√
Stored Procedure	√	√	□	√	×	×	×	×	□	×	×	√	×	×
Inference	√	√	□	√	√	×	√	√	□	√	×	√	√	√
Alternate Encodings	√	√	□	√	√	×	√	×	□	×	×	√	√	√

Table 2: Comparison of SQL injection detection techniques with respect to attack types

Attack Types	Techniques that can detect all attacks of that type (√)	Techniques that can detect the attacks only partially (□)	Techniques that is not able to detect attacks of that type (×)
Tautologies	72%	14%	14%
Piggy-backed	64%	14%	22%
Illegal/ Incorrect	64%	14%	22%
Union	64%	14%	22%
Stored Procedure	29%	14%	57%
Inference	72%	14%	14%
Alternate Encodings	57%	14%	29%

A Systematic Literature Review on SQL Injection Attacks Techniques and Common Exploited Vulnerabilities

-

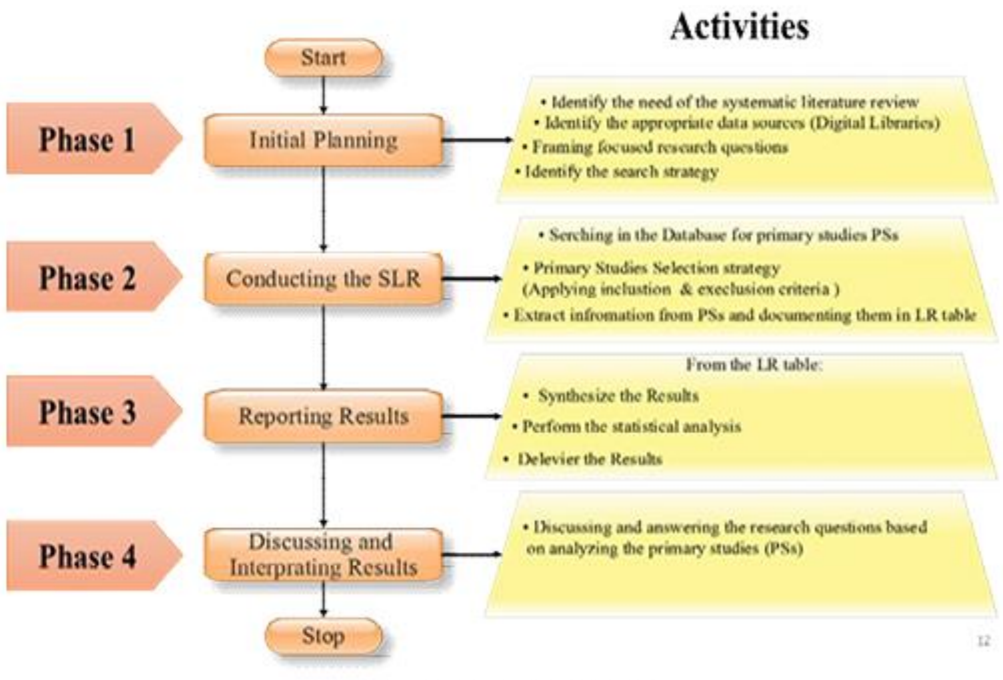
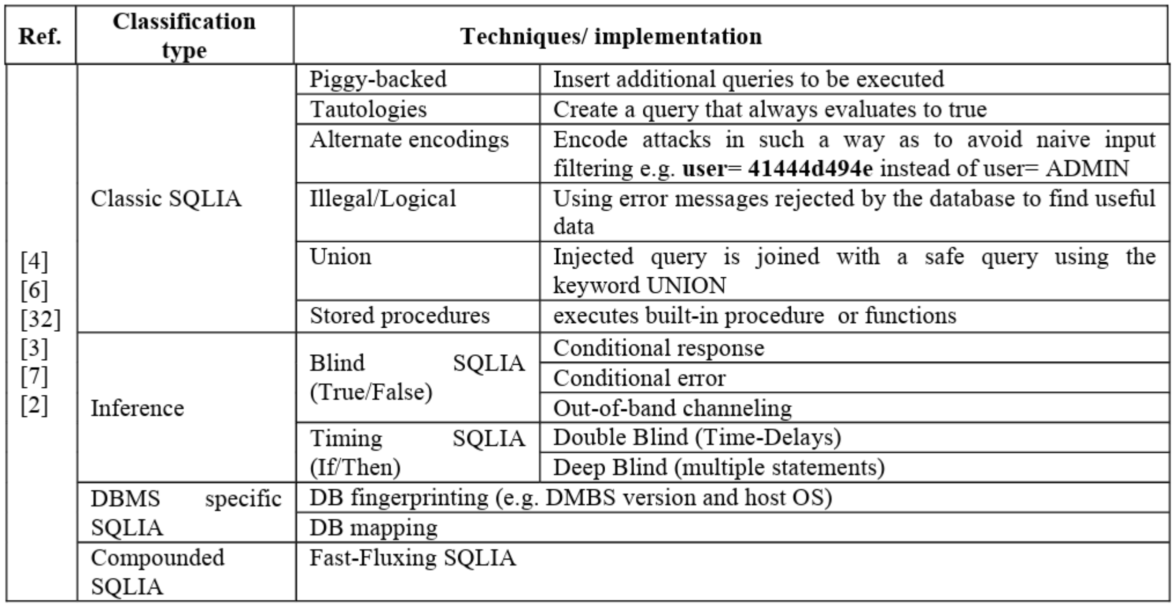

Activities
Start
Phase 1
Initial Planning
• Identify the need of the systematic literature review
• Identify the appropriate data sources (Digital Libraries)
• Framing focused research questions
• Identify the search strategy
Phase 2
Conducting the SLR
• Serching in the Database for primary studies PSs
• Primary Studies Selection strategy (Applying inclustion & execlusion criteria)
• Extract infromation from PSs and documenting them in LR table
Phase 3
Reporting Results
From the LR table:
• Synthesize the Results
• Perfom the statistical analysis
• Delevier the Results
Phase 4
Discussing and Interprating Results
• Discussing and answering the research questions based on analyzing the primary studies (PSs)
Stop
12

Fig. 1. Research Phases

summarize SQLIA based on the mechanisms of attacks



Ref.	Classification type	Techniques/ implementation	
[4] [6] [32] [3] [7] [2]	Classic SQLIA	Piggy-backed	Insert additional queries to be executed
		Tautologies	Create a query that always evaluates to true
		Alternate encodings	Encode attacks in such a way as to avoid naive input filtering e.g. **user= 41444d494e** instead of user= ADMIN
		Illegal/Logical	Using error messages rejected by the database to find useful data
		Union	Injected query is joined with a safe query using the keyword UNION
		Stored procedures	executes built-in procedure or functions
	Inference	Blind SQLIA (True/False)	Conditional response
			Conditional error
			Out-of-band channeling
		Timing SQLIA (If/Then)	Double Blind (Time-Delays)
			Deep Blind (multiple statements)
	DBMS specific SQLIA	DB fingerprinting (e.g. DMBS version and host OS)	
		DB mapping	
	Compounded SQLIA	Fast-Fluxing SQLIA	

Sql INJECTION DETECTION USING HYBRID MODEL

Methodology 4:-

The data is labelled (0 = valid SQL query, 1 = malicious SQL query). It is taken from kaggle website. It contains the following 2 fields:

1. label: 0 = valid SQL query, 1 = malicious SQL query

2. sentence: the text of the SQL query

Implementation:

We have used various supervised Learning Algorithms and to improve the accuracy stacking methods are used.

Supervised learning can be used to classify and process data using machine learning. For which we need labelled data, for which we already know whether the query is malicious or not, this dataset is then used to train the model. After the training is done through various algorithms then this model can be used on unlabeled data for classification of queries. Following approaches are used -

(1) Linear discriminant Analysis – It picks a new dimension such that it maximizes separation between means of projected classes and minimize variance within each

projected class. For multiple variables, similar properties are calculated over the multivariate Gaussian. The statistical properties are then estimated from the data.

(2) Logistic regression - Logistic regression algorithm is used both for classification as well as regression problems using a set of independent variables i.e. we have only two possible scenarios—either the text is plain text or it is a malicious text.

(3) KNN classifier - It is a memory based classification algorithm. The steps are as follows-

First the K-most suitable instances to the data that is being tested are identified. Then suitable labels are extracted. Labels for data being tested are predicted by combining the data being tested

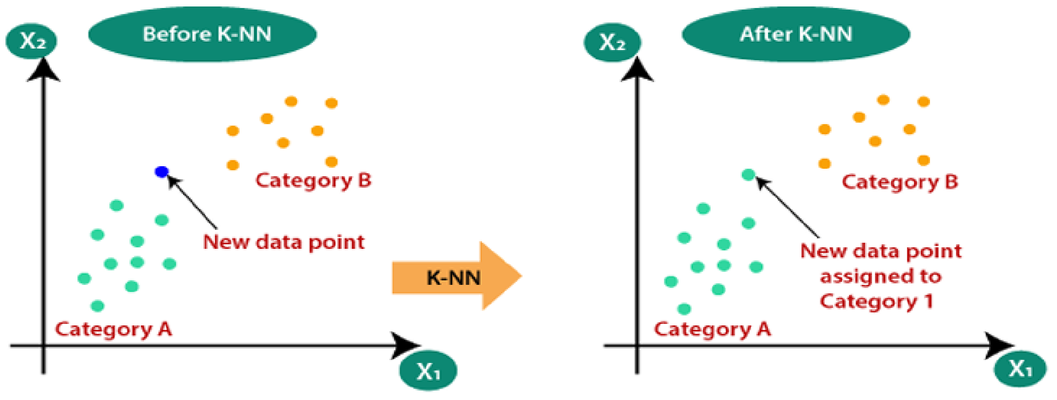
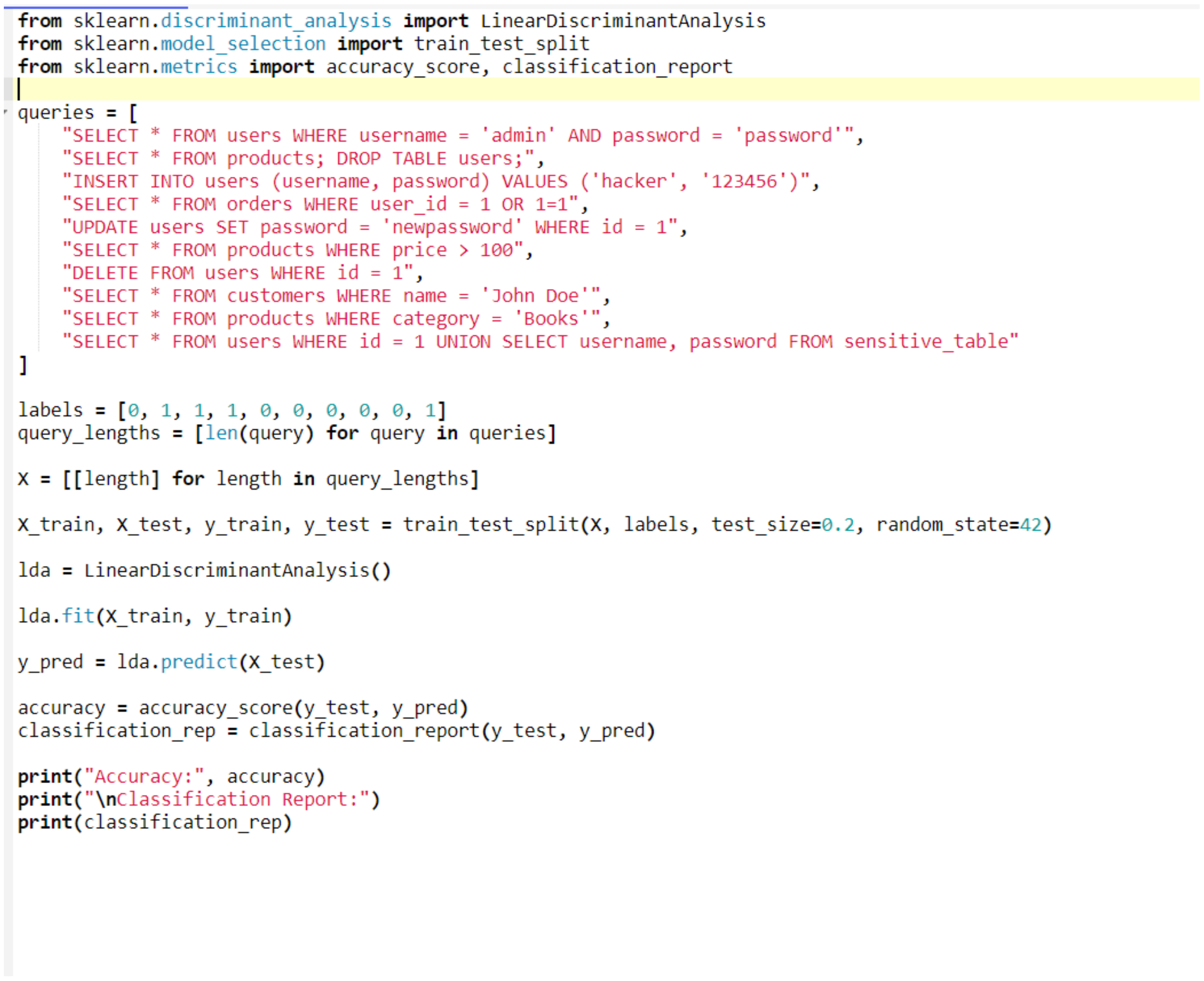


Figure 1: KNN Illustration

Successfully detected SQL injection attacks using the following methods and obtained the following results-

(1) Linear discriminant analysis



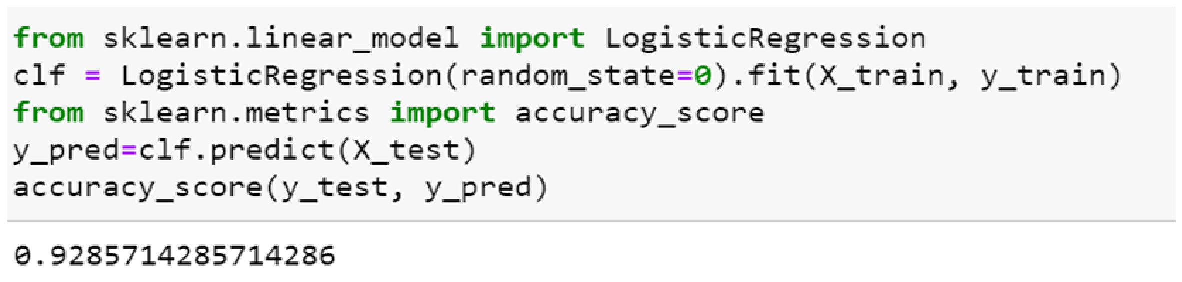
```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

queries = [
    "SELECT * FROM users WHERE username = 'admin' AND password = 'password'",
    "SELECT * FROM products; DROP TABLE users;",
    "INSERT INTO users (username, password) VALUES ('hacker', '123456')",
    "SELECT * FROM orders WHERE user_id = 1 OR 1=1",
    "UPDATE users SET password = 'newpassword' WHERE id = 1",
    "SELECT * FROM products WHERE price > 100",
    "DELETE FROM users WHERE id = 1",
    "SELECT * FROM customers WHERE name = 'John Doe'",
    "SELECT * FROM products WHERE category = 'Books'",
    "SELECT * FROM users WHERE id = 1 UNION SELECT username, password FROM sensitive_table"
]

labels = [0, 1, 1, 1, 0, 0, 0, 0, 0, 1]
query_lengths = [len(query) for query in queries]

X = [[length] for length in query_lengths]

X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2, random_state=42)

lda = LinearDiscriminantAnalysis()

lda.fit(X_train, y_train)

y_pred = lda.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nClassification Report:")
print(classification_rep)
```

OUTPUT:-

Output:

```
Accuracy: 0.5

Classification Report:
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         1
           1       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

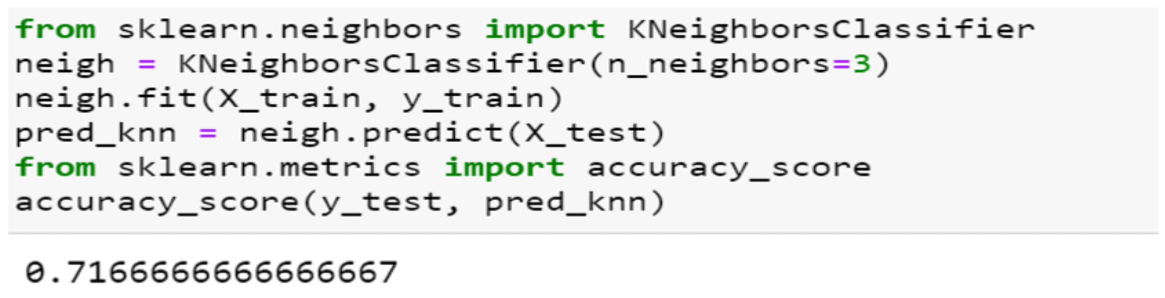
2)Logistic regression



```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
from sklearn.metrics import accuracy_score
y_pred=clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.9285714285714286
```

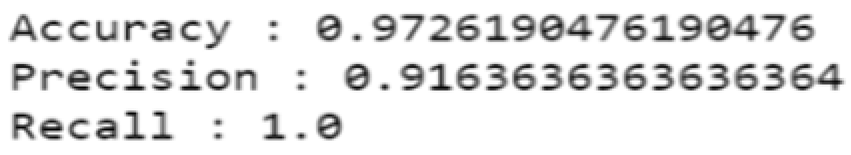
3)KNN classifier:-



```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
pred_knn = neigh.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_test, pred_knn)
```

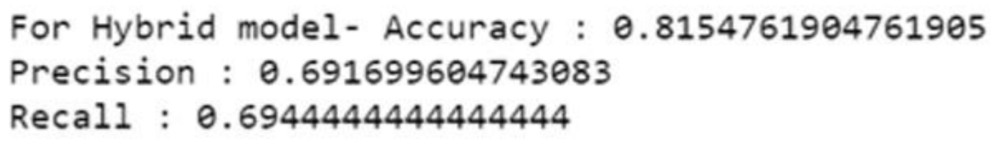
```
0.7166666666666667
```

(4) Neural Network



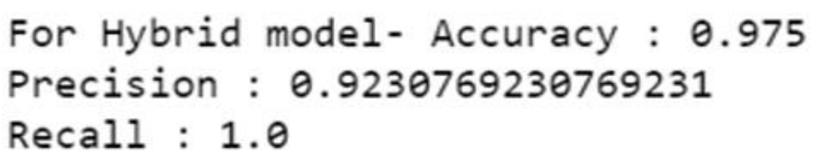
```
Accuracy : 0.9726190476190476
Precision : 0.9163636363636364
Recall : 1.0
```

(5) Stacking Algorithm



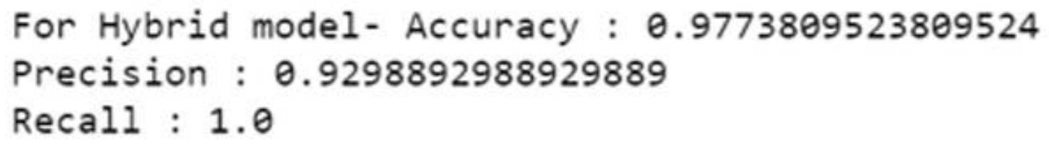
```
For Hybrid model- Accuracy : 0.8154761904761905
Precision : 0.691699604743083
Recall : 0.6944444444444444
```

Performance metric when Base model - KNN and Meta model - LR



```
For Hybrid model- Accuracy : 0.975
Precision : 0.9230769230769231
Recall : 1.0
```

Performance metric when Base Model - LR, KNN and Meta model – LR

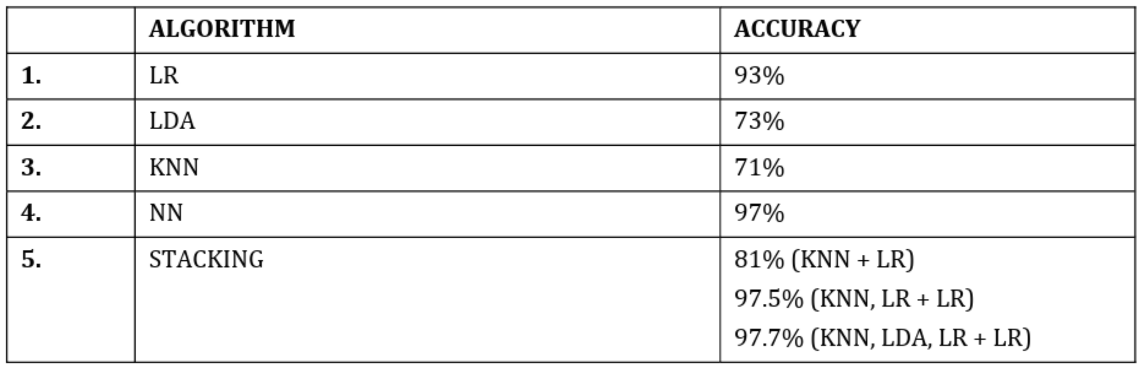


```
For Hybrid model- Accuracy : 0.9773809523809524
Precision : 0.9298892988929889
Recall : 1.0
```

Performance metric when Base Model - LR, KNN, LDA and Meta model – LR

We implemented three machine learning approaches along with neural networks (deep learning) to detect SQL injection attacks. Accuracy of each approach is-

Table 1: Comparison of accuracy of models



	ALGORITHM	ACCURACY
1.	LR	93%
2.	LDA	73%
3.	KNN	71%
4.	NN	97%
5.	STACKING	81% (KNN + LR) 97.5% (KNN, LR + LR) 97.7% (KNN, LDA, LR + LR)

Systematic literature review on SQL revention attacks:-

Methodology 5:

Context-sensitive string evaluation :-

Context-sensitive string evaluation is a method used to interpret and process strings based on the specific context in which they appear. It involves analyzing the content of the string and applying rules, grammar, or predefined conditions to derive meaning or perform specific actions. The evaluation takes into account factors such as the programming language, domain, or application being used. By considering the context, the string can be interpreted as variable names, function calls, mathematical expressions, database queries, or other relevant entities. This approach allows for flexible and dynamic handling of strings, enabling the execution of different operations based on the specific context in which they are encountered.

SQL Injection Attack :-

SQLPIA stands for SQL Injection Attack (SQLIA). It is a type of security vulnerability and attack that targets web applications utilizing databases. SQL injection attacks occur when an attacker manipulates user-supplied input to inject malicious SQL commands into the application's database query. This can lead to unauthorized access, data theft, data manipulation, or even control over the entire database. SQLPIA exploits the improper handling or validation of user input, allowing attackers to execute arbitrary SQL commands and bypass security measures. Preventing SQLPIA requires implementing proper input sanitization, parameterized queries, and strict validation techniques to ensure the integrity and security of the application's database interactions. Regular security audits, vulnerability assessments, and secure coding practices are essential to protect against SQLPIA and maintain robust web application security.

Implementation:

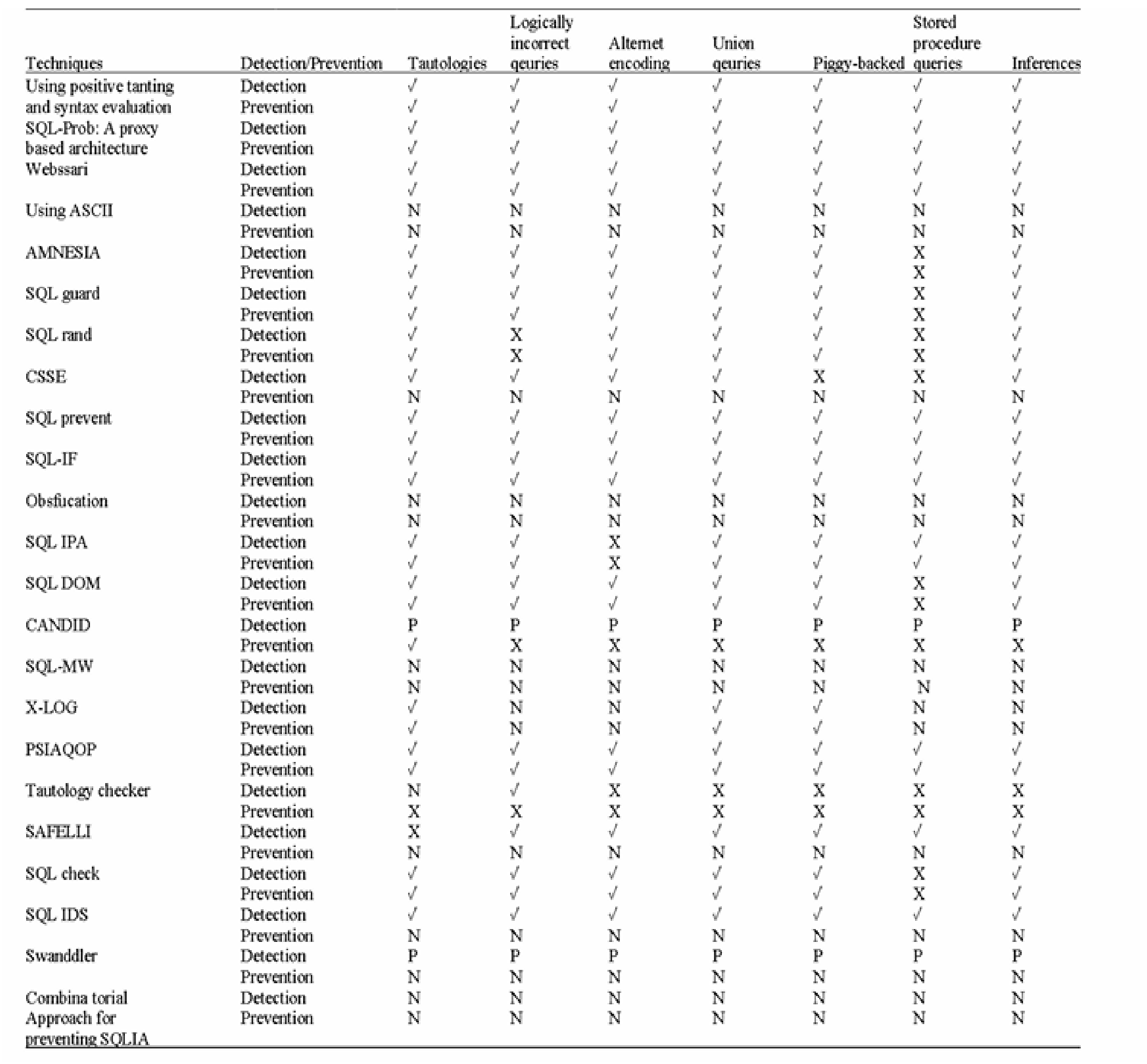
tautology checker:-

A tautology checker is a computational tool or algorithm used to analyze logical expressions and determine whether they are tautologies. A tautology is a statement that is inherently true, regardless of the truth values of its individual components. The tautology checker works by evaluating the logical expression and checking if it remains true for all possible truth value assignments to its variables. It applies logical rules, such as De Morgan's laws, double negation, and truth table analysis, to determine the validity of the expression. Tautology checkers are commonly used in fields such as logic, mathematics, computer science, and formal reasoning to verify the validity of logical arguments, simplify expressions, or identify contradictions. They provide a valuable tool for ensuring the consistency and correctness of logical reasoning.

SQL-IDs :-

SQL-IDs refer to the unique identifiers assigned to database objects within a SQL (Structured Query Language) database system. In SQL, every object, such as tables, views, indexes, or stored procedures, is assigned a specific identifier known as an SQL-ID. These identifiers serve as a way to uniquely identify and reference the objects within the database. SQL-IDs are often automatically generated by the database system and can be used in SQL statements to perform various operations, such as querying, updating, or deleting data. By using SQL-IDs, developers and administrators can efficiently manage and manipulate the database objects, ensuring proper data organization and retrieval.

Evaluation of detection and detection techniques:



Techniques	Detection/Prevention	Tautologies	Logically incorrect qeuries	Altemet encoding	Union qeuries	Piggy-backed	Stored procedure queries	Inferences
Using positive tanting	Detection	√	√	√	√	√	√	√
and syntax evaluation	Prevention	√	√	√	√	√	√	√
SQL-Prob: A proxy	Detection	√	√	√	√	√	√	√
based architecture	Prevention	√	√	√	√	√	√	√
Webssari	Detection	√	√	√	√	√	√	√
	Prevention	√	√	√	√	√	√	√
Using ASCII	Detection	N	N	N	N	N	N	N
	Prevention	N	N	N	N	N	N	N
AMNESIA	Detection	√	√	√	√	√	X	√
	Prevention	√	√	√	√	√	X	√
SQL guard	Detection	√	√	√	√	√	X	√
	Prevention	√	√	√	√	√	X	√
SQL rand	Detection	√	X	√	√	√	X	√
	Prevention	√	X	√	√	√	X	√
CSSE	Detection	√	√	√	√	X	X	√
	Prevention	N	N	N	N	N	N	N
SQL prevent	Detection	√	√	√	√	√	√	√
	Prevention	√	√	√	√	√	√	√
SQL-IF	Detection	√	√	√	√	√	√	√
	Prevention	√	√	√	√	√	√	√
Obsfucation	Detection	N	N	N	N	N	N	N
	Prevention	N	N	N	N	N	N	N
SQL IPA	Detection	√	√	X	√	√	√	√
	Prevention	√	√	X	√	√	√	√
SQL DOM	Detection	√	√	√	√	√	X	√
	Prevention	√	√	√	√	√	X	√
CANDID	Detection	P	P	P	P	P	P	P
	Prevention	√	X	X	X	X	X	X
SQL-MW	Detection	N	N	N	N	N	N	N
	Prevention	N	N	N	N	N	N	N
X-LOG	Detection	√	N	N	√	√	N	N
	Prevention	√	N	N	√	√	N	N
PSIAQOP	Detection	√	√	√	√	√	√	√
	Prevention	√	√	√	√	√	√	√
Tautology checker	Detection	N	√	X	X	X	X	X
	Prevention	X	X	X	X	X	X	X
SAFELLI	Detection	X	√	√	√	√	√	√
	Prevention	N	N	N	N	N	N	N
SQL check	Detection	√	√	√	√	√	X	√
	Prevention	√	√	√	√	√	X	√
SQL IDS	Detection	√	√	√	√	√	√	√
	Prevention	N	N	N	N	N	N	N
Swanddler	Detection	P	P	P	P	P	P	P
	Prevention	N	N	N	N	N	N	N
Combina torial	Detection	N	N	N	N	N	N	N
Approach for	Prevention	N	N	N	N	N	N	N
preventing SQLIA								

Prevention

Methodology 1:

A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm.

1.Cross-site scripting (XSS) attacks:

2.SQL injection attacks:

SQL-injection attacks could be in six categories:

a)Boolean-based SQL injection or tautology attack:

Boolean values (True or False) are used to carry out this type of SQL injection. The malicious SQL query forces the web application to return a different result depending on whether the query returns a TRUE or FALSE result.

b) Union-based SQL injection: This is the most popular of all the SQL injections. It uses the UNION statement to integrate two or more select statements in a SQL query thereby obtaining data illegally from the database.

c) Error-based SQL injection: This is the simplest of all the SQL injection vulnerabilities; however, it only affects web applications that use MS-SQL Server. The most common form of this vulnerability requires an attacker to supply an SQL statement with improper input causing a syntax error such as providing a string when the SQL query is expecting an integer.

d) Batch query SQL injection/piggy backing attacks: This form of injection is dangerous as it attempts to take full control of the database. An attacker terminates the original query of the application and injects his own query into the database server.

e) Like-based SQL injection: This injection type is used by hackers to impersonate a particular user using the SQL keyword LIKE with a wildcard operator (%).

f) Hexadecimal/decimal/binary variation attack (encoded injection): In this type of injection, the hacker leverages on the diversity of the SQL language by using hexadecimal or decimal representations of the keywords instead of the regular strings and characters of the injection code.

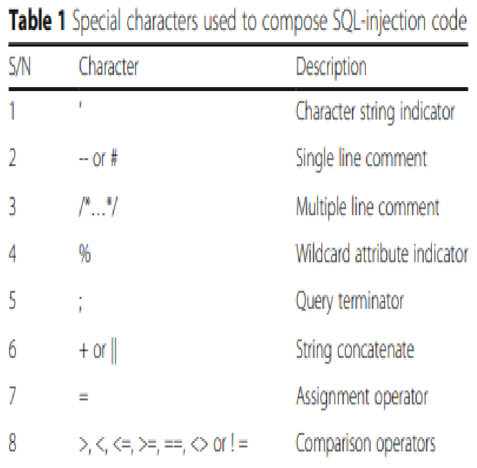


Table 1 Special characters used to compose SQL-injection code

S/N	Character	Description
1	'	Character string indicator
2	-- or #	Single line comment
3	/*...*/	Multiple line comment
4	%	Wildcard attribute indicator
5	;	Query terminator
6	+ or ||	String concatenate
7	=	Assignment operator
8	>, <, <=, >=, ==, <> or ! =	Comparison operators

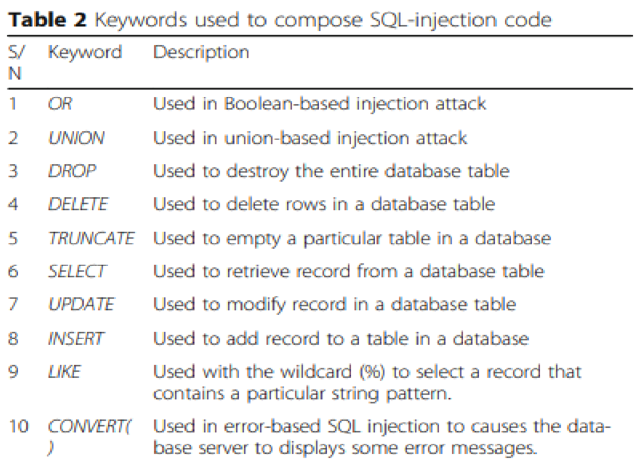


Table 2 Keywords used to compose SQL-injection code

S/ N	Keyword	Description
1	*OR*	Used in Boolean-based injection attack
2	*UNION*	Used in union-based injection attack
3	*DROP*	Used to destroy the entire database table
4	*DELETE*	Used to delete rows in a database table
5	*TRUNCATE*	Used to empty a particular table in a database
6	*SELECT*	Used to retrieve record from a database table
7	*UPDATE*	Used to modify record in a database table
8	*INSERT*	Used to add record to a table in a database
9	*LIKE*	Used with the wildcard (%) to select a record that contains a particular string pattern.
10	*CONVERT( )*	Used in error-based SQL injection to causes the data-base server to displays some error messages.

Implementation Techniques:

The proposed detection and prevention technique:

1.**Formation of SQL injection string patterns:** Every form of attacks has certain characters and keywords that hackers do manipulate to perpetuate their attacks.

2.Designing parse tree for the various forms of attacks:

Parse tree was used to represent the syntactic pattern of the various forms of SQL-Injection and Cross Site Scripting attacks. The parse trees are as follows:

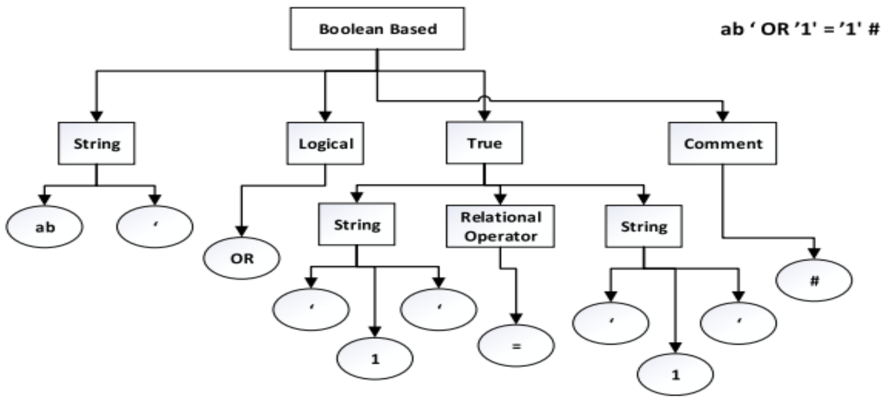


Fig. 1 Parse tree to depict Boolean-based SQL injection attacks. (ii). Union-based SQL injection string

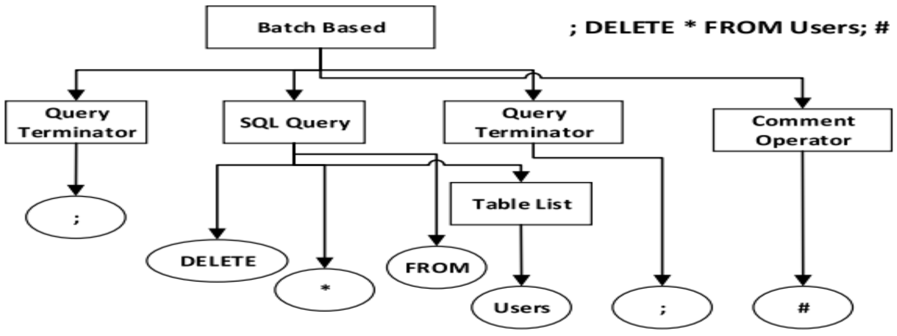


Fig. 4 Parse tree to depict SQL injection attacks using batch query. (ii). Like-based injection attack

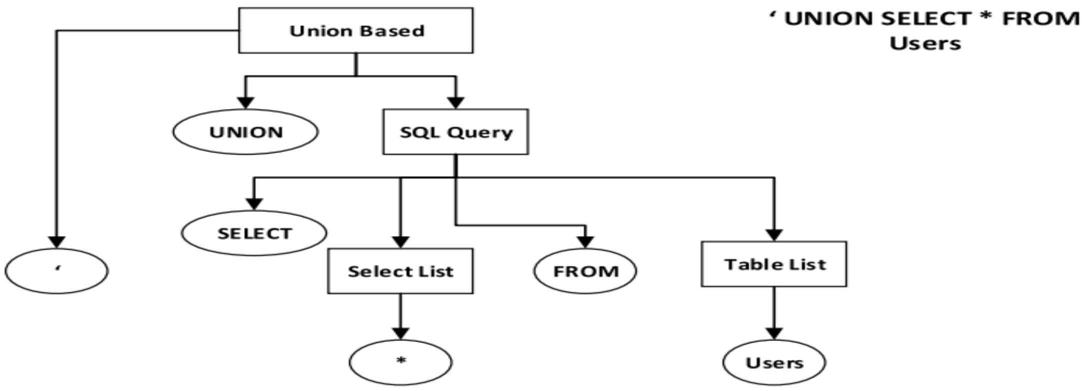


Fig. 2 A parse tree to depict union-based SQL injection attacks. (ii). Error-based SQL injection string

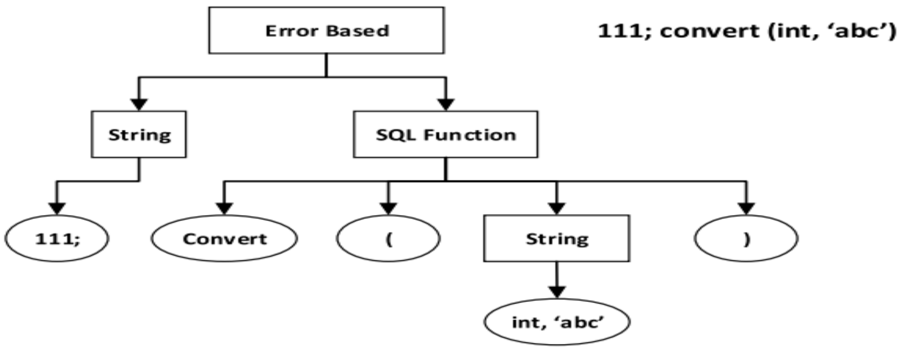


Fig. 3 Parse tree to depict error-based SQL injection attacks. (ii). Batch query SQL injection attacks

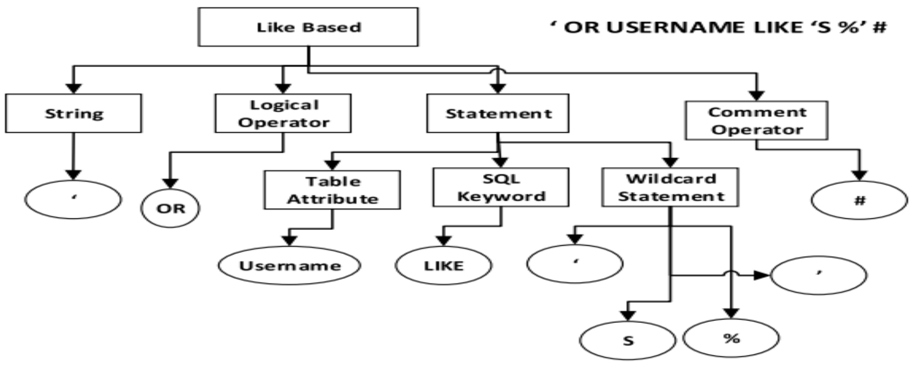


Fig. 5 Parse tree to depict like-based injection attacks. (ii). XSS injection attacks

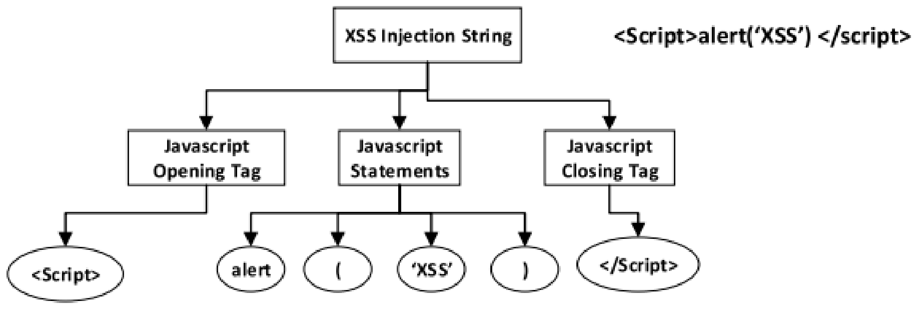


Fig. 6 Parse tree to depict XSS injection attacks

2.3 Preventing SQL-injection and XSS attacks using KMP algorithm:

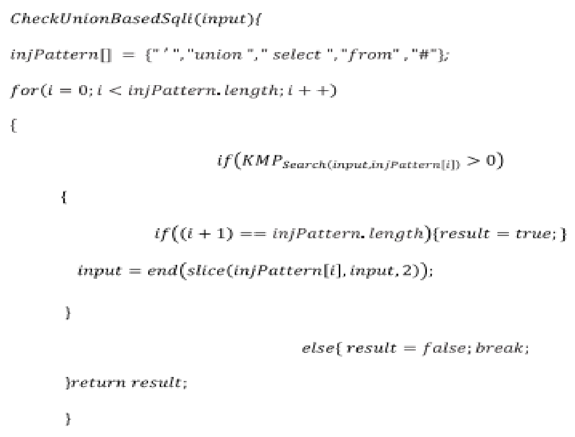
KMP string matching algorithm was used to compare user's input string with different SQL injection and XSS attacks patterns that have been formulated. The algorithm goes thus:

```
$I = \sum_{i=0}^{n} f_i$    Where $f$ is the user's input from each form text field
$filter(I)$ { $data = convertASCIItoString(I)$;
        $if(data <> " ")${
                $a = checkBooleanBasedSqli(data)$
                $b = checkUnionBasedSqli(data)$;
                $c = checkErrorBasedSqli(data)$;
                $d = checkBatchQuerySqli(data)$;
                $e = checkLikeBasedSqli(data)$;
                $f = checkXss(data)$;
                    $if(true(a||b||c||d||e||f))${
                      $blockUser()$;
                      $resetHTTP()$;
                      $warningMessage()$;
                    }
                      $else$ {$grantAccess()$;}
        }
```

Formulating the filter functions:

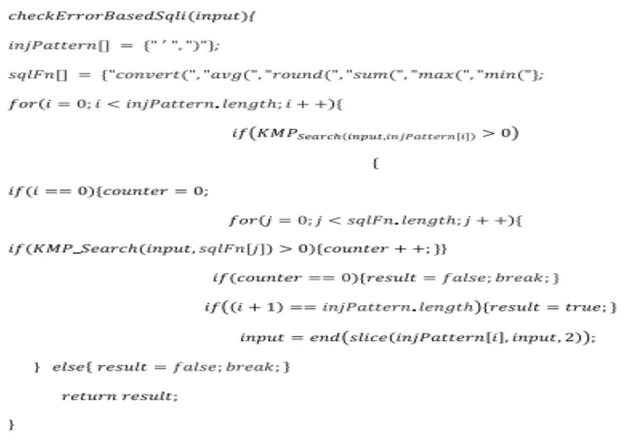
(i). Formulating the checkBooleanBasedSqli( ) function: this was used to prevent Boolean-based SQL injection attack

(ii). Formulating the checkUnionBasedSqli() function: this was used to prevent union-based SQL injection attack:



```
CheckUnionBasedSqli(input){
injPattern[] = {" ' ","union "," select ","from" ,"#"};
for(i = 0; i < injPattern.length; i + +)
{
                            if(KMP_Search(input,injPattern[i]) > 0)
    {
                   if((i + 1) == injPattern.length){result = true; }
     input = end(slice(injPattern[i], input, 2));
    }
                                  else{ result = false; break;
    }return result;
    }
```

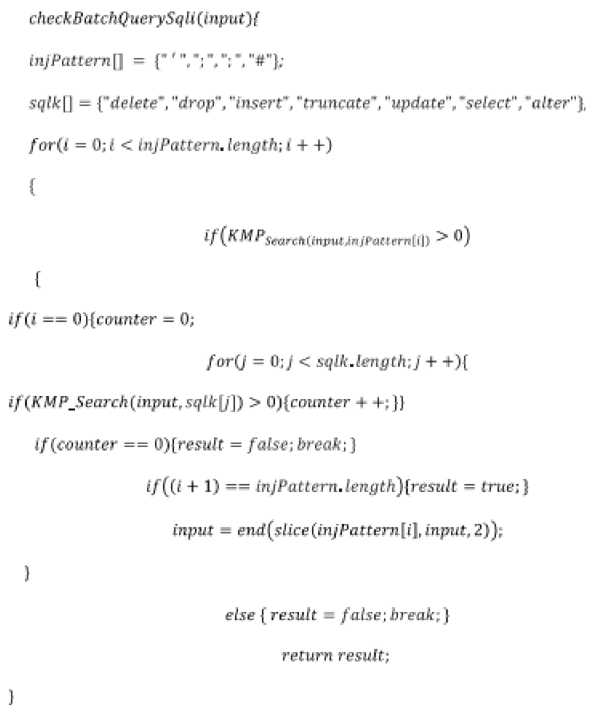
(i)



```
checkErrorBasedSqli(input){
injPattern[] = {" ' ",")"};
sqlFn[] = {"convert(","avg(","round(","sum(","max(","min("};
for(i = 0; i < injPattern.length; i + +){
                              if(KMP_Search(input,injPattern[i]) > 0)
                                          {
if(i == 0){counter = 0;
                              for(j = 0; j < sqlFn.length; j + +){
if(KMP_Search(input, sqlFn[j]) > 0){counter + +; }}
                          if(counter == 0){result = false; break; }
                         if((i + 1) == injPattern.length){result = true; }
                              input = end(slice(injPattern[i], input, 2));
   }  else{ result = false; break; }
        return result;
}
```

(ii)

(iii)Formulating the checkBatchQuerySqli() function: this was used to prevent batch query SQL injection attack:



```
   checkBatchQuerySqli(input){
   injPattern[] = {" ' ",";",";","#"};
   sqlk[] = {"delete","drop","insert","truncate","update","select","alter"},
   for(i = 0; i < injPattern.length; i + +)
   {
                           if(KMP_Search(input,injPattern[i]) > 0)
    {
if(i == 0){counter = 0;
                           for(j = 0; j < sqlk.length; j + +){
if(KMP_Search(input, sqlk[j]) > 0){counter + +; }}
    if(counter == 0){result = false; break; }
                    if((i + 1) == injPattern.length){result = true; }
                       input = end(slice(injPattern[i], input, 2));
  }
                            else { result = false; break; }
                                 return result;
}
```

(iv)Formulating the checkLikeBasedSqlis() function: this was used to prevent like-based SQL injection attack:

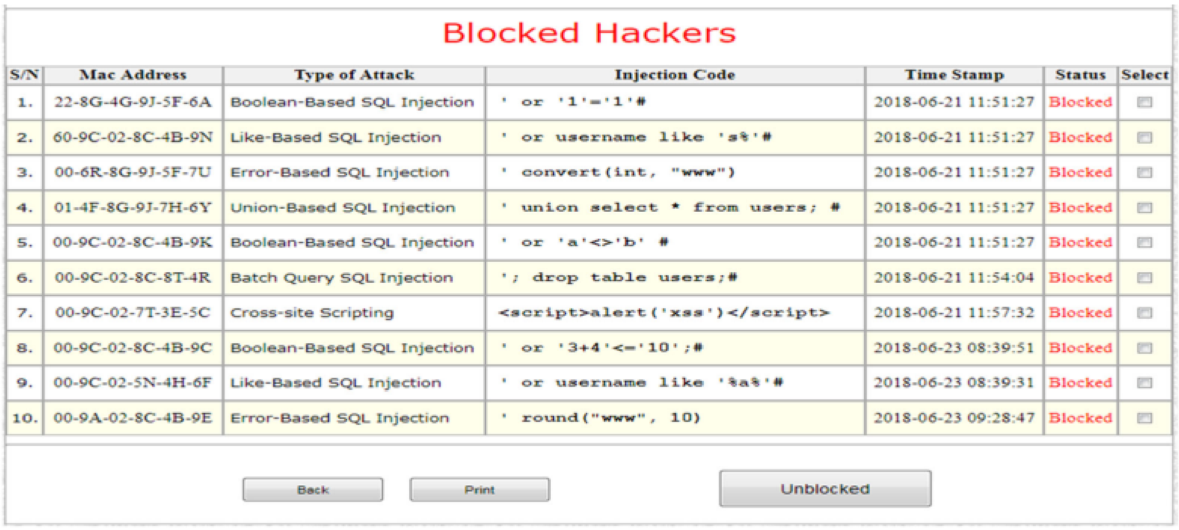
```
checkBatchQuerySqli(input){
injPattern[] = {" ' ",";",";","#"};
sqlk[] = {"delete","drop","insert","truncate","update","select","alter'
for(i = 0; i < injPattern.length; i++)
{
            if($KMP_{Search(input,injPattern[i])}$ > 0)
 {
if(i == 0){counter = 0;
            for(j = 0; j < sqlk.length; j++){
if(KMP_Search(input, sqlk[j]) > 0){counter++;}}
  if(counter == 0){result = false; break;}
        if((i + 1) == injPattern.length){result = true;}
          input = end(slice(injPattern[i], input, 2));
}
              else { result = false; break;}
                return result;
```

```
checkLikeBasedSqli(input){
injPattern[] = {" ' ","like "," ' ","%" " ' ","#"};
lOprt[] = {"or","||"};
for(i = 0; i < injPattern.length; i++)
{
            if($KMP_{Search(input,injPattern[i])}$ > 0)
                 {
if(i == 0){counter = 0;
            for(j = 0; j < lOprt.length; j++)
 {
         if($KMP_{Search(input,lOprt[j])}$ > 0){counter++;}
    }
          if(counter == 0) {result = false; break;}
        if((i + 1) == injPattern.length){result = true;}
          input = end(slice(injPattern[i], input, 2));
   } else  { result = false; break;}
                return result;
```

The Test Plan:

Table 4 The test plan

S/N	Attack type	Sample injection code
1	Boolean-based SQLi	' OR " = "; #
2	Boolean-based SQLi	' OR '1'='1'; #
3	Boolean-based SQLi	' OR '3'! ='8' ;#
4	Boolean-based SQLi	' OR 'a'<>'b' ;#
5	Boolean-based SQLi	aa' OR '2 + 3' < = '7' ;#
6	Like-based SQLi	a' OR username LIKE 'S%';#
7	Like-based SQLi	' OR password LIKE '%2%';#
8	Like-based SQLi	' OR username LIKE '%e';#
9	Union-based SQLi	'UNION select * from users; #
10	Union-based SQLi	'UNION select cardNo, pin from customer; #
11	Error-based SQLi	' convert(int, (select * from users LIMIT 1))
12	Error-based SQLi	' convert(int, "aaaa")
13	Error-based SQLi	' round((select username from users), 3)
14	Batch query SQLi	' ; drop table users ; #
15	Batch query SQLi	' ; delete * from customer ; #
16	Batch query SQLi	' ; insert into users values ('Bala', '1234') ; #
17	Batch query SQL injection	' ; update table users set username = 'Bala', password ='123' ; #
18	Encoded cross-site scripting	<script> alert(&#34; XSS &#34;) </script>
19	Encoded SQL injection	& # *x*39 & # *x*85 & # *x*78 & # *x*73 & # *x*79 & # *x*78 & # *x*32 & # *x*83 & # *x*69 & # *x*76 & # *x*69 & # *x*67 & # *x*84 & # *x*32 & # *x* 42 & # *x*32 & # *x*70 & # *x*82 & # *x*79 & # *x*77 & # *x*32 & # *x*117 & # *x*115 & # *x*101 & # *x*114 & # *x*115 & # *x*45 & # *x*45
20	Cross-site scripting	<script> alert('XSS') </script>
21	Cross-site scripting	<script>myFunction( );</script>



Blocked Hackers

S/N	Mac Address	Type of Attack	Injection Code	Time Stamp	Status	Select
1.	22-8G-4G-9J-5F-6A	Boolean-Based SQL Injection	' or '1'='1'#	2018-06-21 11:51:27	Blocked	☐
2.	60-9C-02-8C-4B-9N	Like-Based SQL Injection	' or username like 's%'#	2018-06-21 11:51:27	Blocked	☐
3.	00-6R-8G-9J-5F-7U	Error-Based SQL Injection	' convert(int, "www")	2018-06-21 11:51:27	Blocked	☐
4.	01-4F-8G-9J-7H-6Y	Union-Based SQL Injection	' union select * from users; #	2018-06-21 11:51:27	Blocked	☐
5.	00-9C-02-8C-4B-9K	Boolean-Based SQL Injection	' or 'a'<>'b' #	2018-06-21 11:51:27	Blocked	☐
6.	00-9C-02-8C-8T-4R	Batch Query SQL Injection	'; drop table users;#	2018-06-21 11:54:04	Blocked	☐
7.	00-9C-02-7T-3E-5C	Cross-site Scripting	<script>alert('xss')</script>	2018-06-21 11:57:32	Blocked	☐
8.	00-9C-02-8C-4B-9C	Boolean-Based SQL Injection	' or '3+4'<='10';#	2018-06-23 08:39:51	Blocked	☐
9.	00-9C-02-5N-4H-6F	Like-Based SQL Injection	' or username like '%a%'#	2018-06-23 08:39:31	Blocked	☐
10.	00-9A-02-8C-4B-9E	Error-Based SQL Injection	' round("www", 10)	2018-06-23 09:28:47	Blocked	☐

Fig. 8 Attack detection interface showing the record of blocked hackers

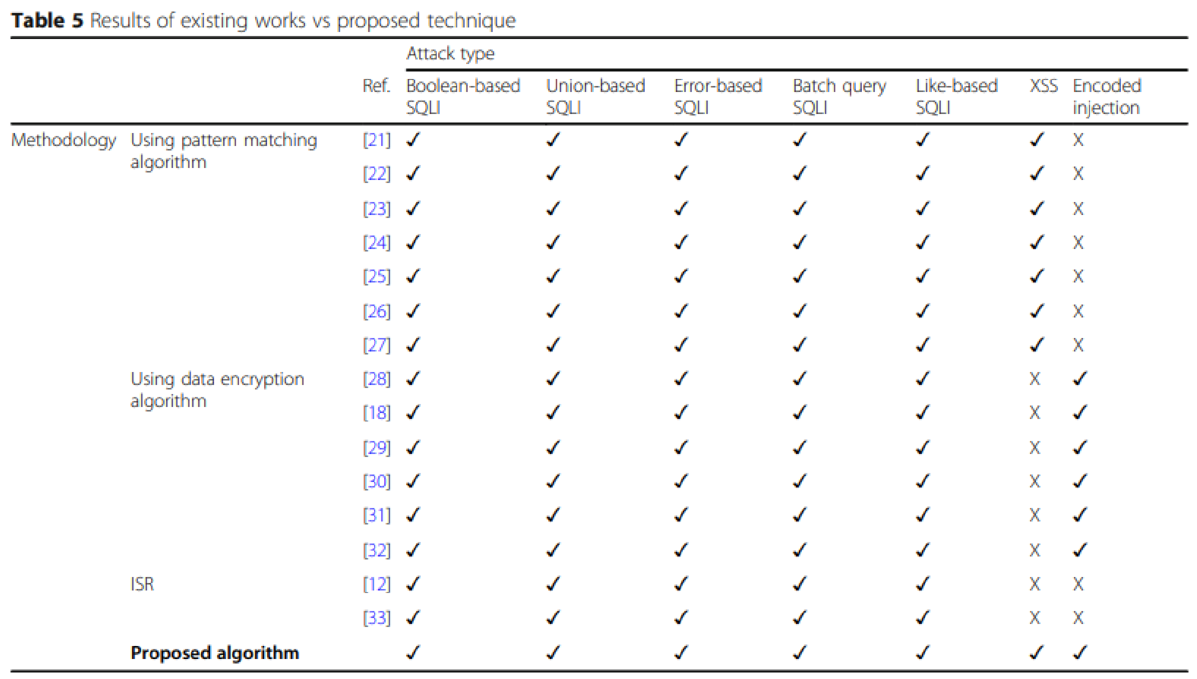


Table 5 Results of existing works vs proposed technique

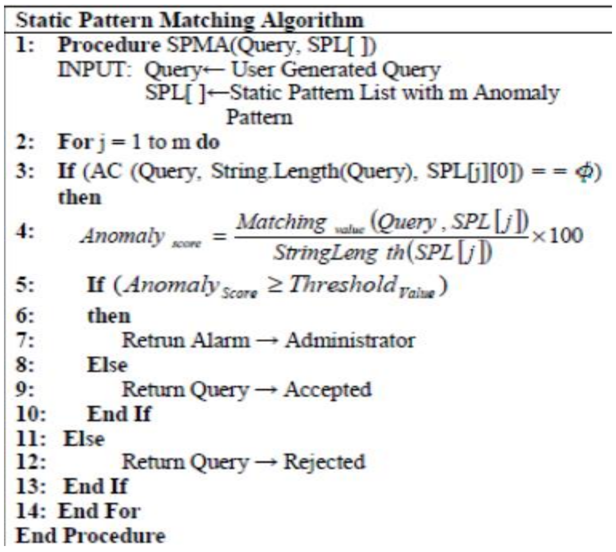
			Attack type						
		Ref.	Boolean-based SQLI	Union-based SQLI	Error-based SQLI	Batch query SQLI	Like-based SQLI	XSS	Encoded injection
Methodology	Using pattern matching algorithm	[21]	✓	✓	✓	✓	✓	✓	X
		[22]	✓	✓	✓	✓	✓	✓	X
		[23]	✓	✓	✓	✓	✓	✓	X
		[24]	✓	✓	✓	✓	✓	✓	X
		[25]	✓	✓	✓	✓	✓	✓	X
		[26]	✓	✓	✓	✓	✓	✓	X
		[27]	✓	✓	✓	✓	✓	✓	X
	Using data encryption algorithm	[28]	✓	✓	✓	✓	✓	X	✓
		[18]	✓	✓	✓	✓	✓	X	✓
		[29]	✓	✓	✓	✓	✓	X	✓
		[30]	✓	✓	✓	✓	✓	X	✓
		[31]	✓	✓	✓	✓	✓	X	✓
		[32]	✓	✓	✓	✓	✓	X	✓
	ISR	[12]	✓	✓	✓	✓	✓	X	X
		[33]	✓	✓	✓	✓	✓	X	X
	Proposed algorithm		✓	✓	✓	✓	✓	✓	✓

Methodology 2:

Preventing SQL injection attack using pattern matching algorithm

Implementation Techniques:

(i)Static Phase:

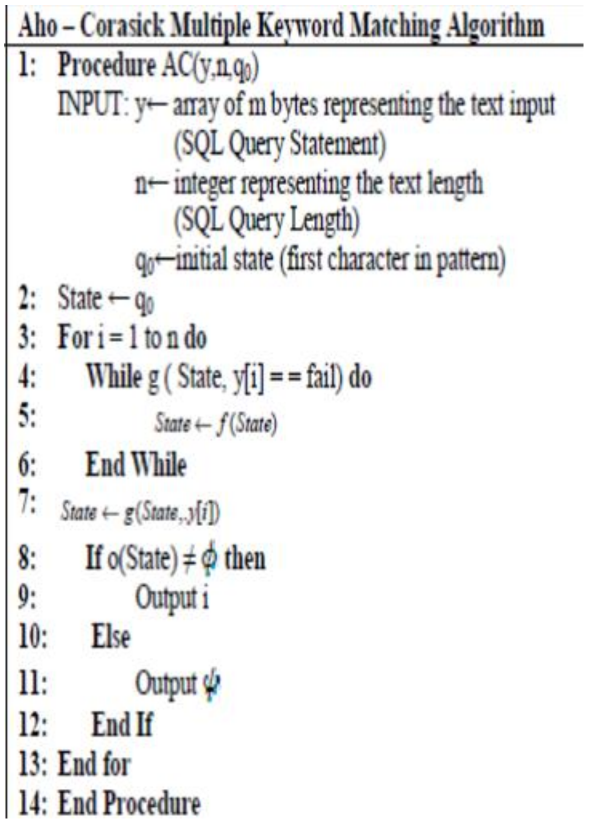


```
Static Pattern Matching Algorithm
1:  Procedure SPMA(Query, SPL[ ])
    INPUT:  Query← User Generated Query
            SPL[ ]←Static Pattern List with m Anomaly
                   Pattern
2:  For j = 1 to m do
3:  If (AC (Query, String.Length(Query), SPL[j][0]) = = $\Phi$)
    then
4:      $Anomaly_{score} = \frac{Matching_{value}(Query, SPL[j])}{StringLength(SPL[j])} \times 100$
5:      If ($Anomaly_{Score} \geq Threshold_{Value}$)
6:      then
7:          Retrun Alarm → Administrator
8:      Else
9:          Return Query → Accepted
10:     End If
11:  Else
12:         Return Query → Rejected
13:  End If
14: End For
End Procedure
```

(ii)Dynamic Phase

A. Anomaly Score value calculation

B. Pattern Matching Algorithm



```
Aho – Corasick Multiple Keyword Matching Algorithm
1:  Procedure AC(y,n,$q_0$)
    INPUT: y← array of m bytes representing the text input
              (SQL Query Statement)
           n← integer representing the text length
              (SQL Query Length)
           $q_0$←initial state (first character in pattern)
2:  State ← $q_0$
3:  For i = 1 to n do
4:     While g ( State, y[i] = = fail) do
5:           $State \leftarrow f(State)$
6:     End While
7:  $State \leftarrow g(State, y[i])$
8:     If o(State) ≠ $\phi$ then
9:          Output i
10:     Else
11:          Output $\phi$
12:     End If
13: End for
14: End Procedure
```

The proposed Architecture:

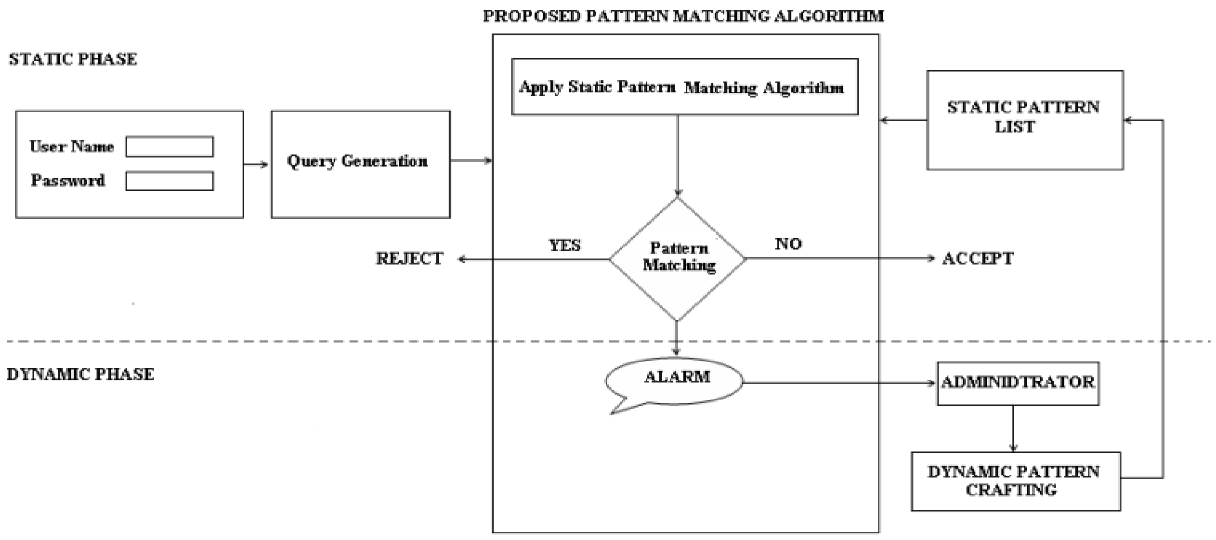


Fig. 1. Proposed Architecture

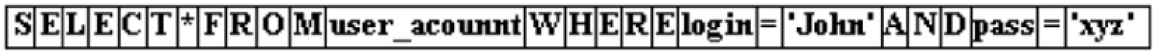

S E L E C T * F R O M user_acounnt W H E R E login = 'John' A N D pass = 'xyz'

Fig. 2. SQL Query Generation with legal user name and password

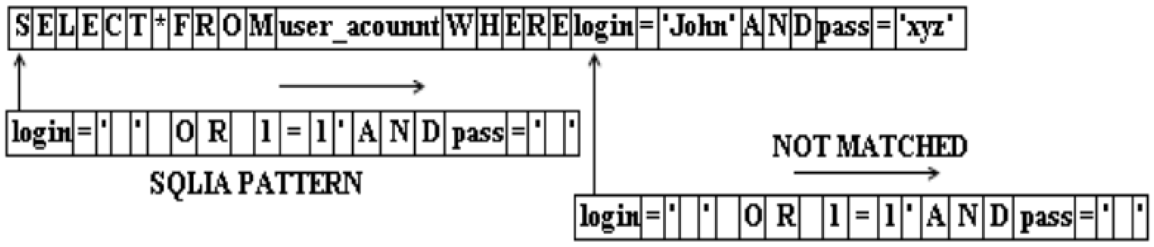

S E L E C T * F R O M user_acounnt W H E R E login = 'John' A N D pass = 'xyz'
login = ' ' O R 1 = 1 ' A N D pass = ' '
SQLIA PATTERN
NOT MATCHED
login = ' ' O R 1 = 1 ' A N D pass = ' '

Fig. 3. SQLIA Pattern Matching Process

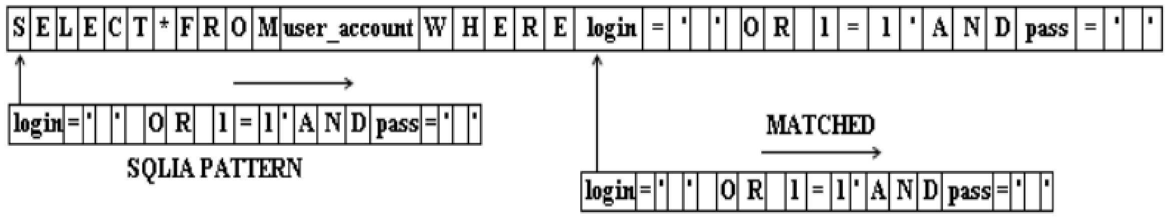

S E L E C T * F R O M user_account W H E R E login = ' ' O R 1 = 1 ' A N D pass = ' '
login = ' ' O R 1 = 1 ' A N D pass = ' '
SQLIA PATTERN
MATCHED
login = ' ' O R 1 = 1 ' A N D pass = ' '

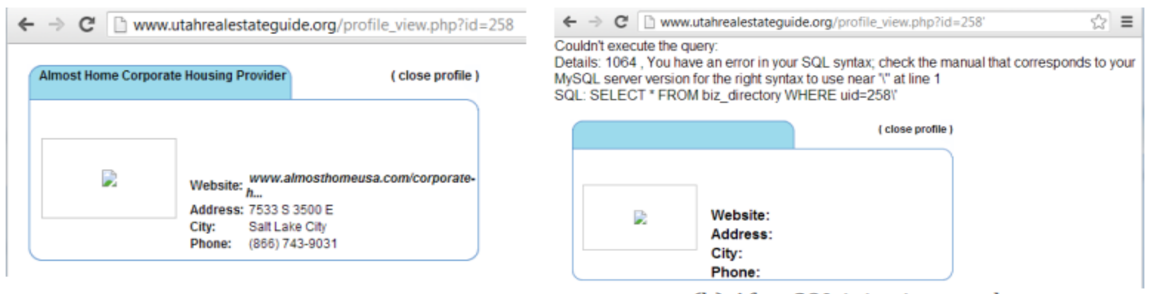
Fig. 4. SQLIA Pattern Exactly Matching

Methodology 3:

SQL Injection Attack Scanner Using Boyer-Moore String Matching Algorithm

1. SQL Injection Attack

2. SQL Injection Methodology



www.utahrealestateguide.org/profile_view.php?id=258
Almost Home Corporate Housing Provider
(close profile)
Website: www.almosthomeusa.com/corporate-h...
Address: 7533 S 3500 E
City: Salt Lake City
Phone: (866) 743-9031
www.utahrealestateguide.org/profile_view.php?id=258'

```
Couldn't execute the query:
Details: 1064 , You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near '\'' at line 1
SQL: SELECT * FROM biz_directory WHERE uid=258\'
```

(close profile)
Website:
Address:
City:
Phone:

(a) Before SQL injection attack (b) After SQL injection attack

Fig. 1. Vulnerable Web page.

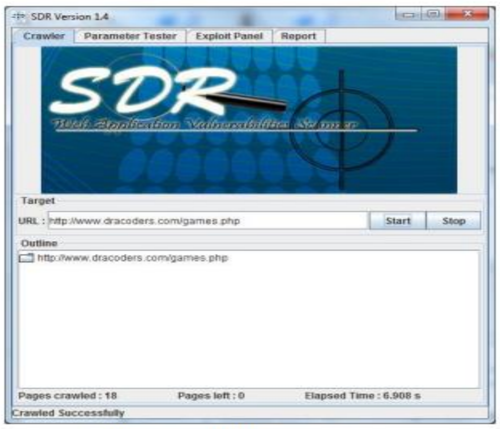
Boyer Moore Algorithm:

The Boyer-Moore string matching algorithm is usually used for searching large amounts of data in a short period of time such as searching for virus patterns and databases. This algorithm is one of the fastest string searching algorithms as it searches the string for the pattern but not each character of the string. As the pattern length increases, the algorithm will run faster

Implementation Techniques:

1. SDR Web Vulnerability Scanner

By using the proposed model, an application to scan web vulnerabilities called Scan-Detect-Report (SDR) has been developed and tested. As shown in fig, at the top of the SDR scanner window, there are four panel reference detection modules as stated earlier in the previous section. To begin the crawling process, a user needs to enter the target URL in the URL text field and click the Start button.



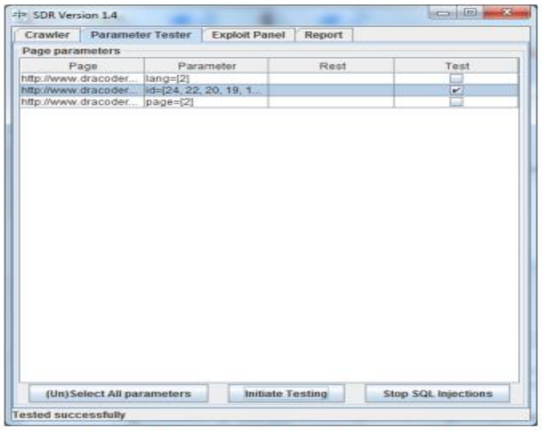
Target
URL : http://www.dracoders.com/games.php

Outline
http://www.dracoders.com/games.php

Pages crawled : 18 Pages left : 0 Elapsed Time : 6.908 s

Crawled Successfully

Fig. 3. Crawler panel.

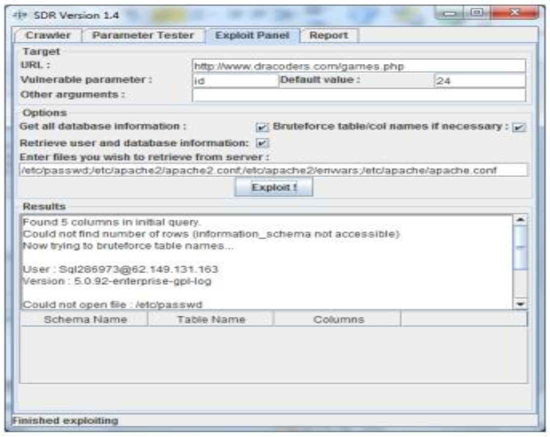


Page parameters

Page	Parameter	Rest	Test
http://www.dracoder...	lang=[2]		☐
http://www.dracoder...	id=[24, 22, 20, 19, 1...		✓
http://www.dracoder...	page=[2]		☐

Tested successfully

Fig. 4. Parameter tester panel.



Target
URL : http://www.dracoders.com/games.php
Vulnerable parameter : id Default value : 24
Other arguments :

Options
Get all database information : ✓ Bruteforce table/col names if necessary : ✓
Retrieve user and database information: ✓
Enter files you wish to retrieve from server :
/etc/passwd;/etc/apache2/apache2.conf;/etc/apache2/envvars;/etc/apache/apache.conf

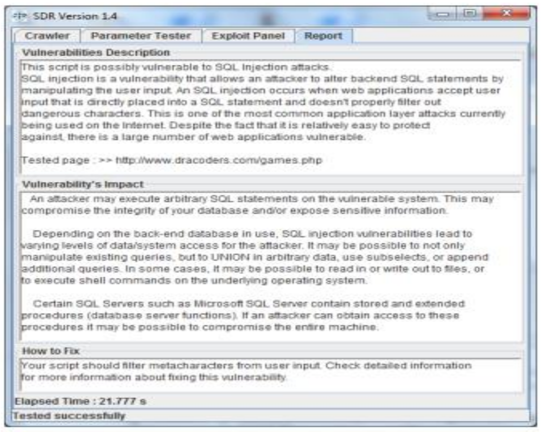
Results

```
Found 5 columns in initial query.
Could not find number of rows (information_schema not accessible)
Now trying to bruteforce table names...

User : Sql286973@62.149.131.163
Version : 5.0.92-enterprise-gpl-log

Could not open file : /etc/passwd
```

Schema Name	Table Name	Columns	

Finished exploiting

Fig. 5. Exploit panel.



Vulnerabilities Description

This script is possibly vulnerable to SQL Injection attacks.
SQL injection is a vulnerability that allows an attacker to alter backend SQL statements by manipulating the user input. An SQL injection occurs when web applications accept user input that is directly placed into a SQL statement and doesn't properly filter out dangerous characters. This is one of the most common application layer attacks currently being used on the Internet. Despite the fact that it is relatively easy to protect against, there is a large number of web applications vulnerable.

Tested page : >> http://www.dracoders.com/games.php

Vulnerability's Impact

An attacker may execute arbitrary SQL statements on the vulnerable system. This may compromise the integrity of your database and/or expose sensitive information.

Depending on the back-end database in use, SQL injection vulnerabilities lead to varying levels of data/system access for the attacker. It may be possible to not only manipulate existing queries, but to UNION in arbitrary data, use subselects, or append additional queries. In some cases, it may be possible to read in or write out to files, or to execute shell commands on the underlying operating system.

Certain SQL Servers such as Microsoft SQL Server contain stored and extended procedures (database server functions). If an attacker can obtain access to these procedures it may be possible to compromise the entire machine.

How to Fix

Your script should filter metacharacters from user input. Check detailed information for more information about fixing this vulnerability.

Elapsed Time : 21.777 s

Tested successfully

Fig. 6. Report panel.

Efficiency and Accuracy:

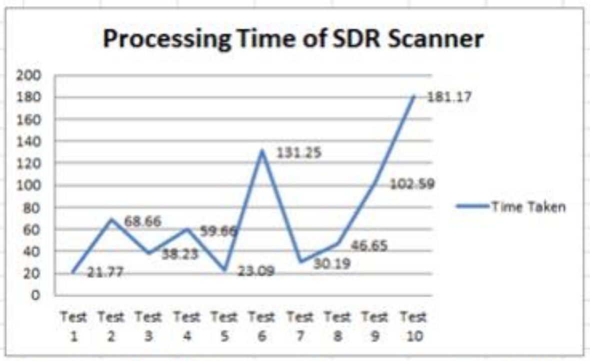
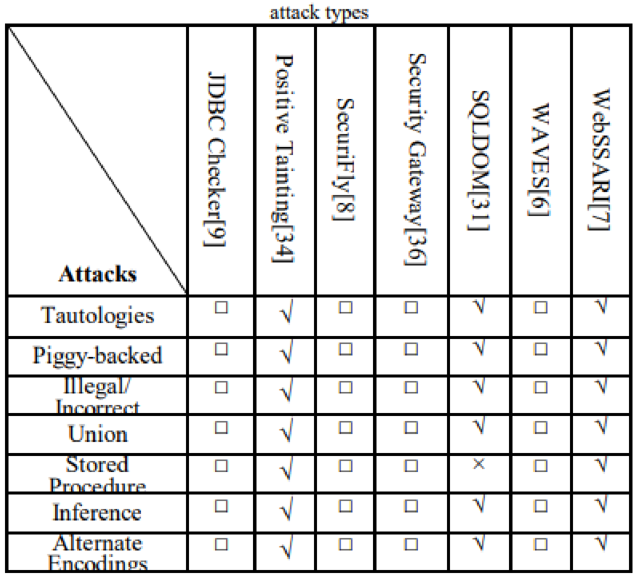


Fig. 7. The efficiency of the proposed model.

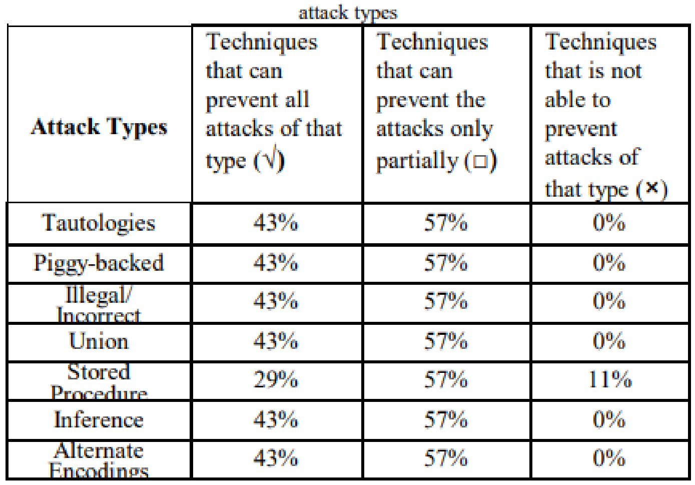
Comparisons:

Table 3: Comparison of SQL injection preventiontechniques with respect to attack types



Attacks	JDBC Checker[9]	Positive Tainting[34]	SecuriFly[8]	Security Gateway[36]	SQLDOM[31]	WAVES[6]	WebSSARI[7]
Tautologies	□	√	□	□	√	□	√
Piggy-backed	□	√	□	□	√	□	√
Illegal/ Incorrect	□	√	□	□	√	□	√
Union	□	√	□	□	√	□	√
Stored Procedure	□	√	□	□	×	□	√
Inference	□	√	□	□	√	□	√
Alternate Encodings	□	√	□	□	√	□	√

Table 4: Comparison of SQL injection prevention techniques with respect to attack types



Attack Types	Techniques that can prevent all attacks of that type (√)	Techniques that can prevent the attacks only partially (□)	Techniques that is not able to prevent attacks of that type (×)
Tautologies	43%	57%	0%
Piggy-backed	43%	57%	0%
Illegal/ Incorrect	43%	57%	0%
Union	43%	57%	0%
Stored Procedure	29%	57%	11%
Inference	43%	57%	0%
Alternate Encodings	43%	57%	0%

Methodology 4 :

The methodology aims to prevent SQL injection attacks using an adaptive algorithm. It combines two techniques: Parse Tree Validation and Code Conversion. The Parse Tree Validation technique involves comparing the parse tree of an SQL statement before and after including user input. If there is a length mismatch, indicating a potential attack, further analysis is performed. In such cases, the user input undergoes code conversion using techniques like ASCII, binary, or hexadecimal. A counter is utilised to differentiate between converted and non-converted inputs. By applying these techniques, the algorithm detects and mitigates SQL injection vulnerabilities effectively.

Implementation :

The proposed method consists of the best features of both parse tree validation technique and code conversion method. In this method we parse the user input and check whether its vulnerable, if there is any chance of vulnerability present then code conversion will be applied over that input. In this way, we can detect and prevent SQL Injection using a single code. Below is the algorithm for the proposed method:

For web pages that saves data to the database:

```
begin()
    get user input
    compare with generalized SQL Query
    if length mismatch
        display – possibly an attack
        newvariable= hexa(user input)
        set counter =1
    else
        display – safe input
        newvariable= user input
        set counter = 0
    display – value to be stored in database 'newvariable'
    end;
```

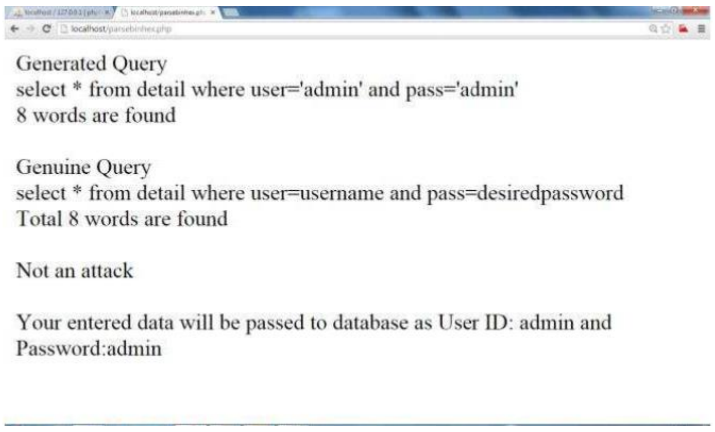
For web pages that only retrieves from the database: begin()

```
    if counter = 0
        display - variable
    else
        variable = ascii(newvariable)
        display - variable
```

end;

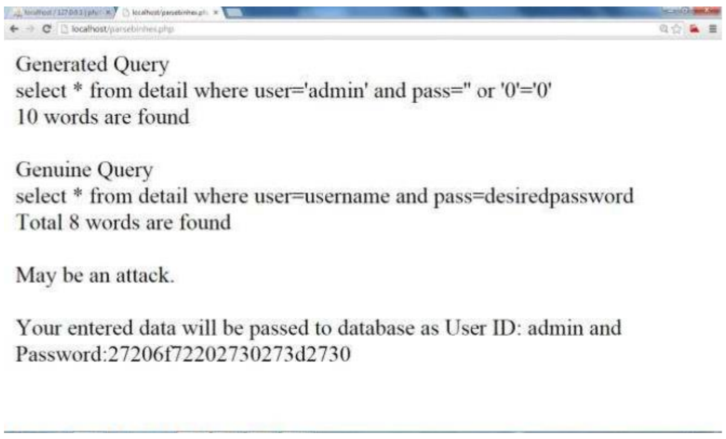
From the value of counter we can come to know whether the user input is converted or not.

After implementing the proposed algorithm in php with mysql, we have obtained a more secured input. We were able to sanitize the user input easily without putting any extra load to the processor or to the database. The figures below show the changes between a safe user



```
Generated Query
select * from detail where user='admin' and pass='admin'
8 words are found

Genuine Query
select * from detail where user=username and pass=desiredpassword
Total 8 words are found

Not an attack

Your entered data will be passed to database as User ID: admin and
Password:admin
```

Safe Input



```
Generated Query
select * from detail where user='admin' and pass='' or '0'='0'
10 words are found

Genuine Query
select * from detail where user=username and pass=desiredpassword
Total 8 words are found

May be an attack.

Your entered data will be passed to database as User ID: admin and
Password:27206f72202730273d2730
```

Vulnerable Input

input and a vulnerable user input.

Methodology 5:

The utilisation of machine learning in preventing SQL injections serves as an effective mechanism for securing databases on the server side. The methodology involves the creation of a server-side component responsible for detecting SQL injection attack patterns in the values passed to the application server before processing client queries.

The system incorporates two levels of security. In the first level, the patterns generated by the context-free grammar (CFG) rules are compared with the patterns associated with SQL attack rules. If there is no match in the first level, the system proceeds to the next level of security. This level utilises the K-means clustering algorithm to cluster patterns in the dataset and classify them into specific clusters, effectively preventing injection attacks.

The system architecture, as shown in Figure 1, illustrates the flow of queries from the client to the web server. The web browser sends queries to the web server, which then passes the user-provided values to the SQL injection detector. The detector checks for SQL injection attacks in the values. If the values are deemed valid, they are directed to the query processor for execution. The processed query is then

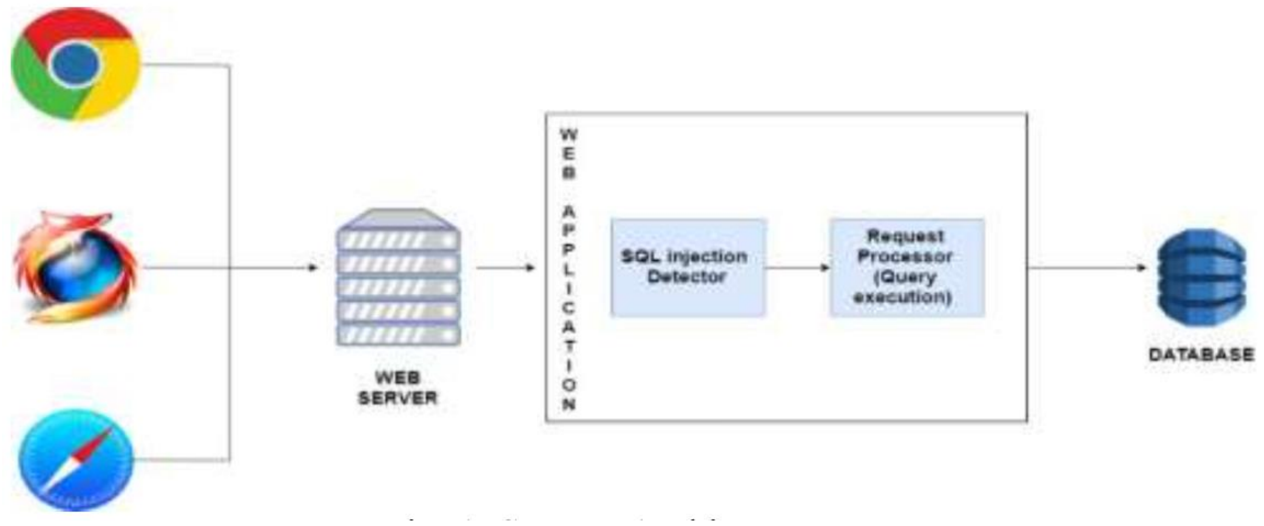


Figure 1 - System Architecture

executed by accessing the database.

Figure 2 presents the functional architecture of the system, providing a comprehensive overview of its operation and the algorithms employed. The user enters data into a form, and the URL intercept engine extracts the values from the query. The injection detector compares the values with predefined patterns using context-free grammar and categorizes attacks using K-means clustering. If no attacks are detected, the user is granted access to the database, and the response is sent back to the client. However, if an attack is identified, access is denied, and the

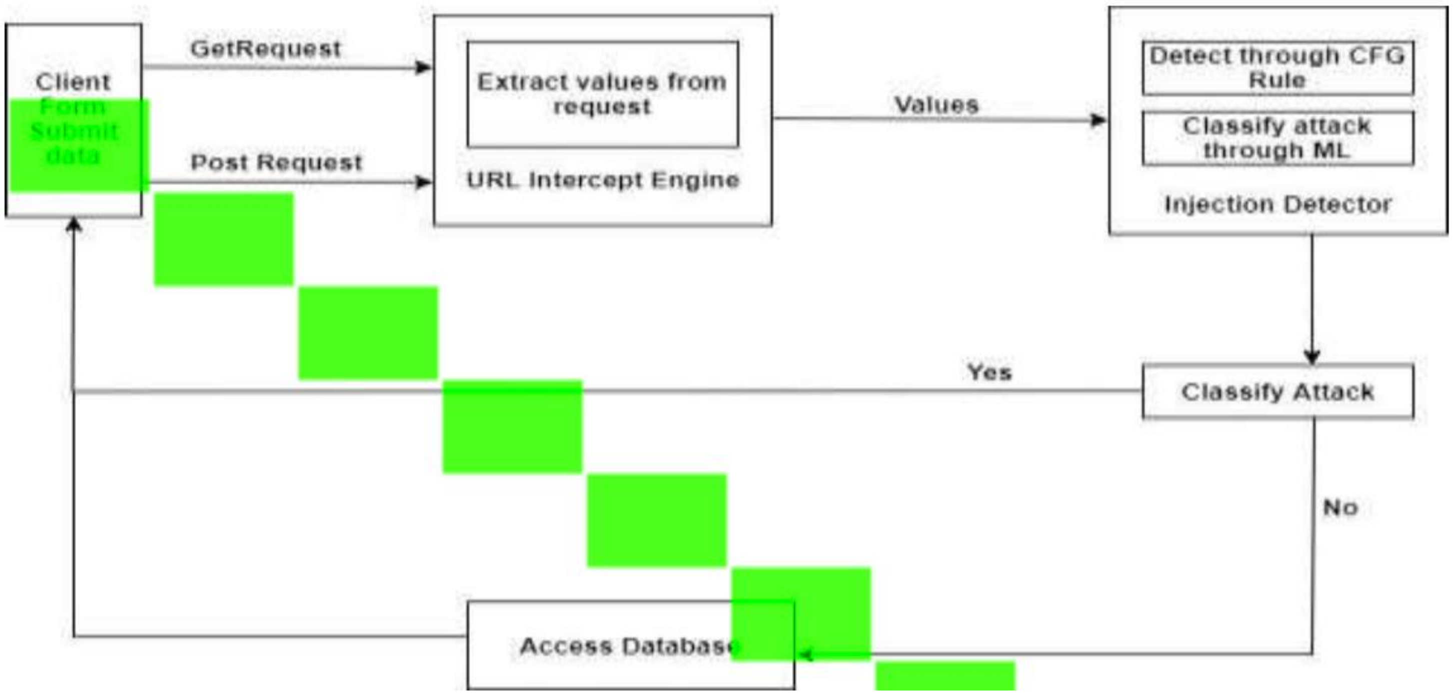
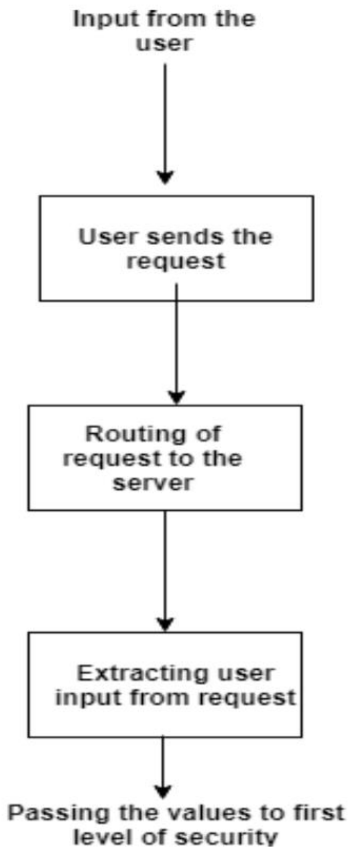


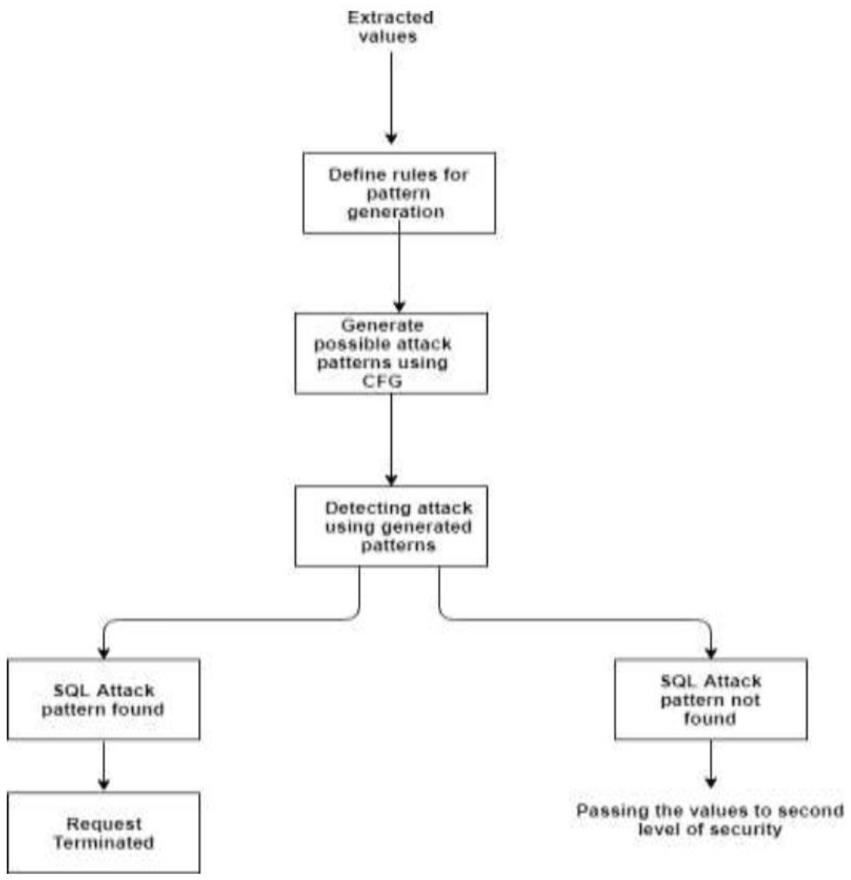
Figure 2 - Functional Architecture

specific type of attack matching the pattern is determined.

The proposed methodology comprises three main steps: the URL Intercept Engine, Context-Free Grammar for SQLi attacks, and pattern classification through machine learning.

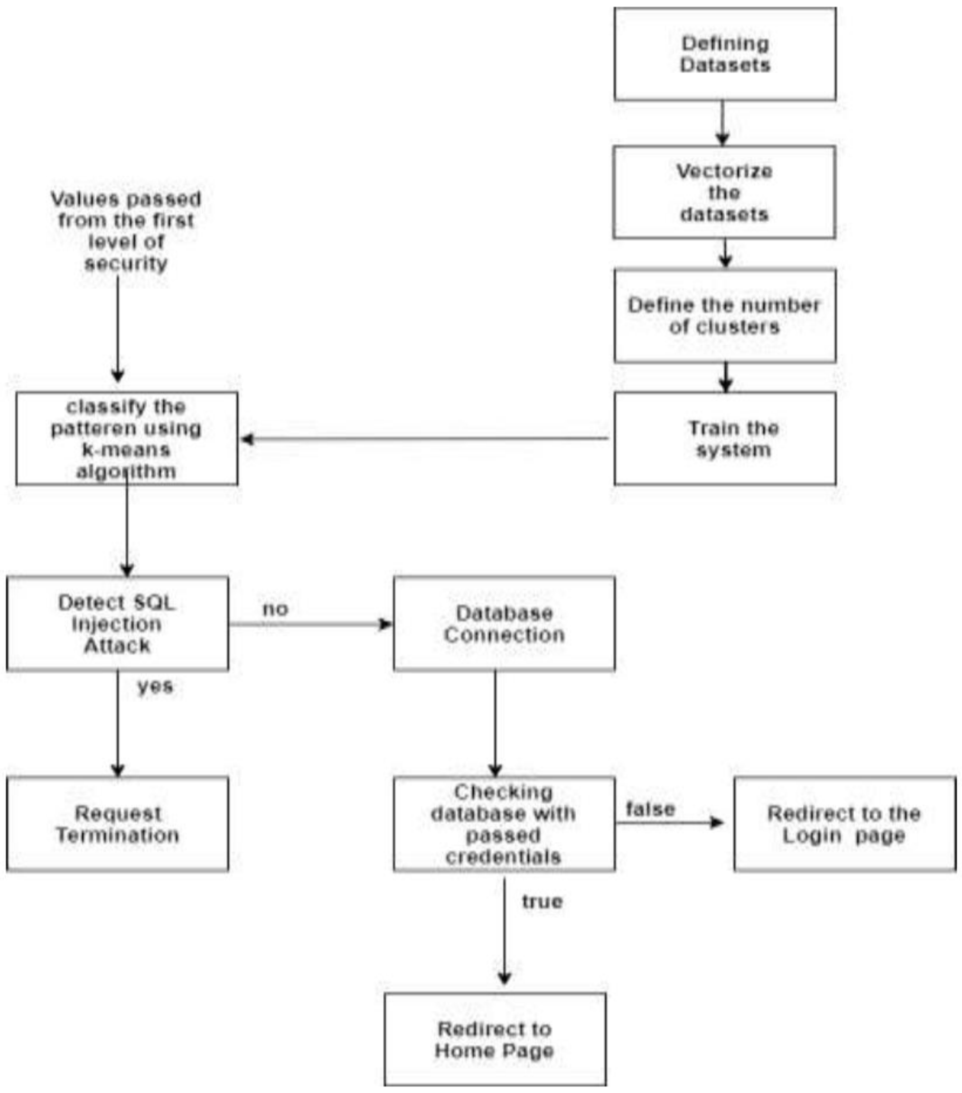
A. URL Intercept Engine : The URL Intercept Engine plays a crucial role in the methodology by identifying and extracting the values from the user's query. It focuses on the vulnerable text fields in web applications, which are commonly targeted by SQL injection attacks. The URL serves as a global address that directs users to specific resources on the World Wide Web, such as web pages, videos, or documents. Attackers with malicious intent may attempt to manipulate the database of an organisation by inserting harmful SQL queries into the text fields. The URL Intercept Engine intercepts the user's query and extracts the values for further analysis and protection against SQL injection attacks.


Input from the user
User sends the request
Routing of request to the server
Extracting user input from request
Passing the values to first level of security


Extracted values
Define rules for pattern generation
Generate possible attack patterns using CFG
Detecting attack using generated patterns
SQL Attack pattern found
Request Terminated
SQL Attack pattern not found
Passing the values to second level of security

B. Context Free Grammar For SQLI Attacks : The next step in the proposed methodology focuses on providing an initial level of security against SQL injection attacks using Context-Free Grammar (CFG). The attack patterns are generated based on the CFG rules specific to SQL injection attacks. The extracted value from the initial segment is compared against these CFG-generated attack patterns. If there is a match between the value and the attack pattern, the system proceeds to the next level of security to further determine whether the value is harmful or not. Even if the value does not match any of the attack patterns created by the CFG, it still undergoes the second level of security. The CFG rules continue to generate patterns until there are no more terminals left.

C. Pattern Classification Through Machine Learning : The pattern classification step contributes the second level of security in the proposed methodology. Figure given below depicts the workflow of pattern classification through machine learning. An unsupervised machine learning algorithm is employed to group different types of attacks into individual clusters based on their fitness values. If the value matches with the cluster associated with any of the identified attack patterns, the system prevents the query from being executed, effectively thwarting the SQL injection attack. On the other hand, if the value does not match any attack pattern cluster, the query is


Defining Datasets
Vectorize the datasets
Define the number of clusters
Train the system
Values passed from the first level of security
classify the patteren using k-means algorithm
Detect SQL Injection Attack
no
yes
Database Connection
Request Termination
Checking database with passed credentials
false
Redirect to the Login page
true
Redirect to Home Page

executed by accessing the database.

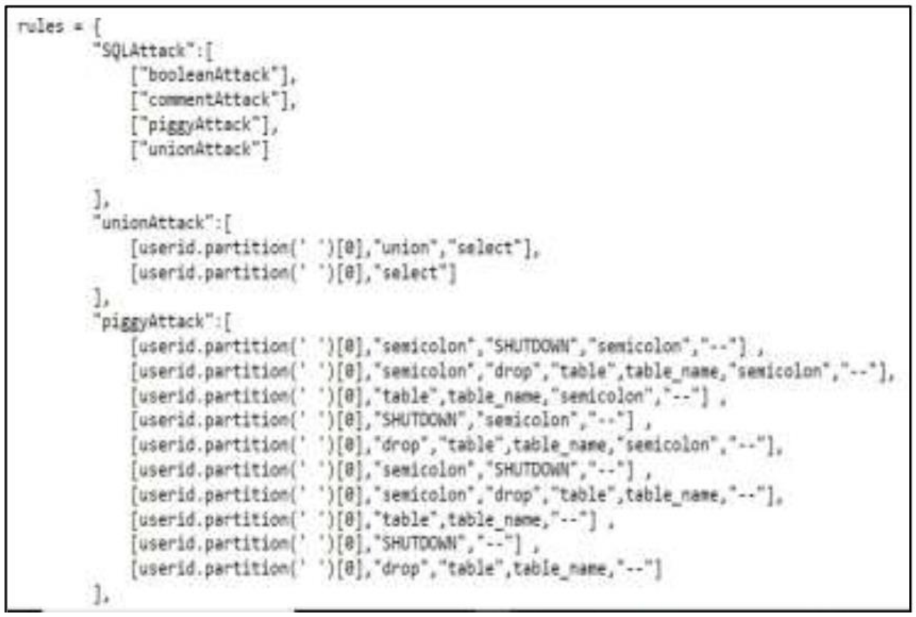
Implementation :

Fig. 3 presents the rules that are defined for generating SQL injection attack patterns to be combined with user input. These rules are utilized by the CFG function responsible for creating the attack pattern. The rules are designed to generate three categories of SQL injection attacks : Boolean Attack, Piggy-Backed Attack, and Union Attack.

Once the user input is extracted, the query is passed to these rules, where the function generates potential attack patterns that can be appended to the user input.

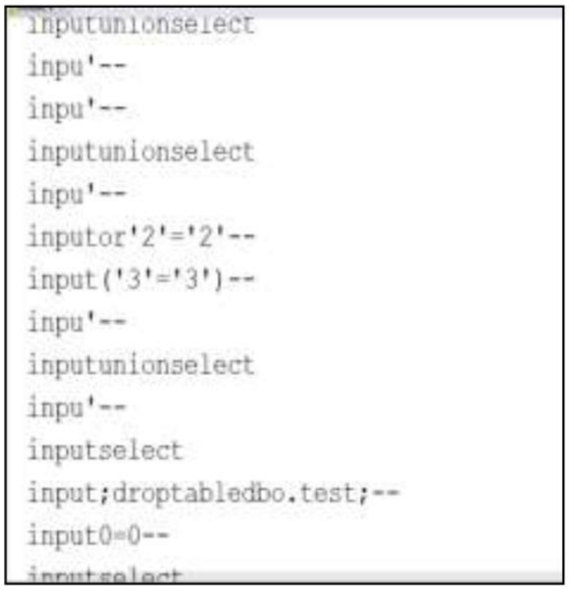
Fig. 4 illustrates the output of the possible attack patterns created by the specified rules shown in Fig. 5. These patterns represent the variations of attacks that can potentially be appended to the end of the user input. The user input and the identified attack patterns are then compared against each other to determine if there is a match or potential SQL injection attack present.

In summary, Fig. 3 displays the rules used for generating SQL injection attack patterns, while Fig. 4 showcases the resulting attack patterns that can be appended to user input. These patterns are compared with the user input to identify and mitigate potential SQL



```
rules = [
        "SQLAttack":[
            ["booleanAttack"],
            ["commentAttack"],
            ["piggyAttack"],
            ["unionAttack"]

        ],
        "unionAttack":[
            [userid.partition(' ')[0],"union","select"],
            [userid.partition(' ')[0],"select"]
        ],
        "piggyAttack":[
            [userid.partition(' ')[0],"semicolon","SHUTDOWN","semicolon","--"] ,
            [userid.partition(' ')[0],"semicolon","drop","table",table_name,"semicolon","--"],
            [userid.partition(' ')[0],"table",table_name,"semicolon","--"] ,
            [userid.partition(' ')[0],"SHUTDOWN","semicolon","--"] ,
            [userid.partition(' ')[0],"drop","table",table_name,"semicolon","--"],
            [userid.partition(' ')[0],"semicolon","SHUTDOWN","--"] ,
            [userid.partition(' ')[0],"semicolon","drop","table",table_name,"--"],
            [userid.partition(' ')[0],"table",table_name,"--"] ,
            [userid.partition(' ')[0],"SHUTDOWN","--"] ,
            [userid.partition(' ')[0],"drop","table",table_name,"--"]
        ],
```

Figure 3 - Rules for Pattern Generation



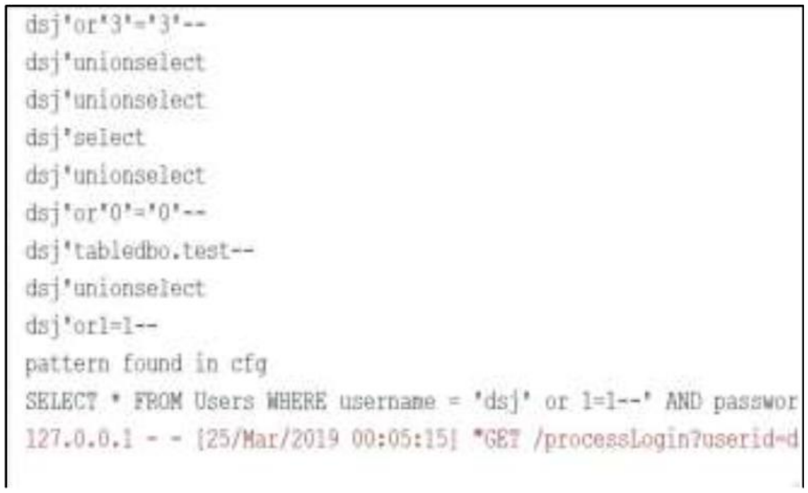
```
inputunionselect
inpu'--
inpu'--
inputunionselect
inpu'--
inputor'2'='2'--
input('3'='3')--
inpu'--
inputunionselect
inpu'--
inputselect
input;droptabledbo.test;--
input0=0--
inputselect
```

Figure 4 - Possible Attack

injection attacks.

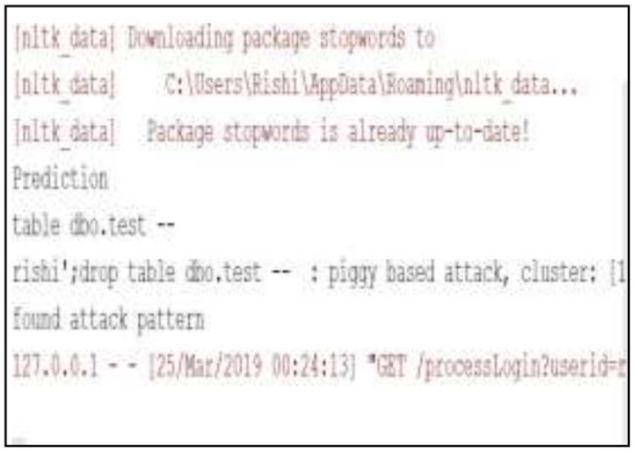
Fig. 5 represents the identification of SQL injection attacks created by the CFG patterns, as well as the creation of the corresponding SQL query to be executed by the SQL server. It encompasses all categories of attacks that can be appended to the end of the username, and ultimately matches with one of the attack patterns generated by the CFG function.

Fig. 6 illustrates the identification of attack patterns by the system itself, which has learned through an unsupervised learning technique using the K-means clustering algorithm. The system leverages the NLTK library to identify and remove stop words such as ‘and,’ ‘or,’ ‘from,' ‘to,’ etc., retaining only the significant words for training the machine learning model. Based on the clustering results, the system identifies the attack pattern and prevents the execution of the query, effectively mitigating the SQL



```
dsj'or'3'='3'--
dsj'unionselect
dsj'unionselect
dsj'select
dsj'unionselect
dsj'or'0'='0'--
dsj'tabledbo.test--
dsj'unionselect
dsj'or1=1--
pattern found in cfg
SELECT * FROM Users WHERE username = 'dsj' or 1=1--' AND passwor
127.0.0.1 - - [25/Mar/2019 00:05:15] "GET /processLogin?userid=d
```

Figure 5 - Identification by the CGF Function



```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Rishi\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Prediction
table dbo.test --
rishi';drop table dbo.test --  : piggy based attack, cluster: [1
found attack pattern
127.0.0.1 - - [25/Mar/2019 00:24:13] "GET /processLogin?userid=r
```

Figure 6 - Identification by Clustering

injection attack.

Methodology 6 :

The experimental analysis of machine learning algorithms in the context of SQL injection attacks (SQLIA) was conducted using the Waikato Environment for Knowledge Analysis (WEKA). The goal was to develop a model that could effectively classify the SQLIA dataset into attack classes, achieving optimal performance. To evaluate the performance of different classification algorithms (supervised learning), two evaluation techniques were employed: hold-out (70%) and 10-fold cross-validation. These techniques were used to assess the algorithms based on various performance metrics, including Kappa Statistic, True Positive (TP) Rate, Accuracy, True Negative (TN), and Training Time (time taken to build the model). The objective was to select the best-performing algorithm based on these criteria.

To assess the performance of the classification algorithms, the model was built using WEKA 3.8.0. The evaluation was conducted using both the hold-out method with 70% of the data allocated for training and the 10-fold cross-validation technique. Several classification algorithms were employed, including Logistic Regression (LRN), SMO, Bayes Network (BNK), IBK, Multilayer Perceptron (MLP), Naive Bayes (NBS), and J48.

During the training process, the performance metrics were measured to determine the effectiveness of each algorithm. The evaluation criteria included correctly classified instances (accuracy), Kappa Statistic, True Positive (TP) Rate, True Negative (TN) Rate, and Training Time (time taken to build the model). These metrics were compared to identify the algorithm that achieved the highest performance based on these criteria.

By evaluating the algorithms using both hold-out and cross-validation techniques, the model's robustness and generalisability were assessed. The comparison of performance metrics allowed for the selection of the most suitable algorithm for effectively classifying SQL injection attacks. This evaluation process provided insights into the strengths and weaknesses of each algorithm and facilitated the decision-making process in choosing the best algorithm for the prevention of SQL injection attacks.

Implementation :

The performance evaluation of the machine learning algorithms was conducted using ten established performance benchmarks, including Accuracy, Kappa Statistic, True Positive (TP) Rate, True Negative (TN) Rate, and Training Time. Figures 1 to provide a comprehensive comparison of the results obtained from the implementation of these algorithms in WEKA.

Choosing the appropriate algorithm for building a model is a critical aspect of machine learning problems. It is essential to consider multiple performance metrics rather than relying solely on a single metric such as accuracy, which is often favoured by researchers. By evaluating the algorithms based on various benchmarks, a more comprehensive assessment of their strengths and weaknesses can be obtained. This approach enables a more informed decision regarding the selection of the optimal algorithm for effectively addressing the problem at hand.

The comparison tables and figures offer insights into the performance of each algorithm across different metrics, facilitating a comprehensive analysis of their capabilities. By considering a broader range of evaluation criteria, researchers can make more informed decisions when selecting the most suitable algorithm for their specific requirements.

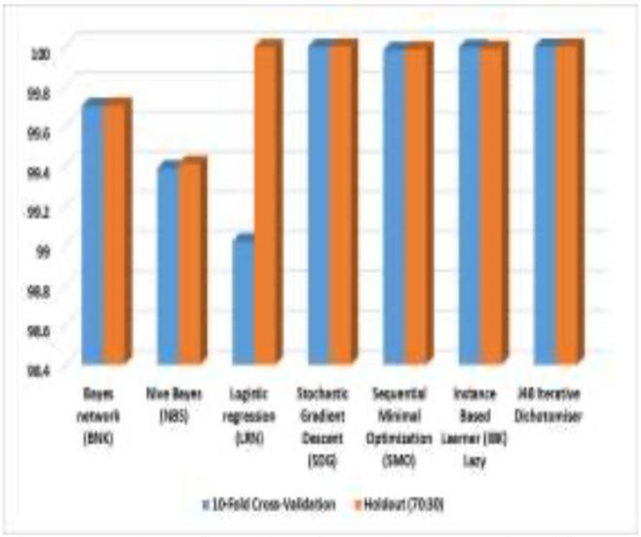


Figure 1 - Comparison based on Accuracy of Hold-Out and Cross-Validation

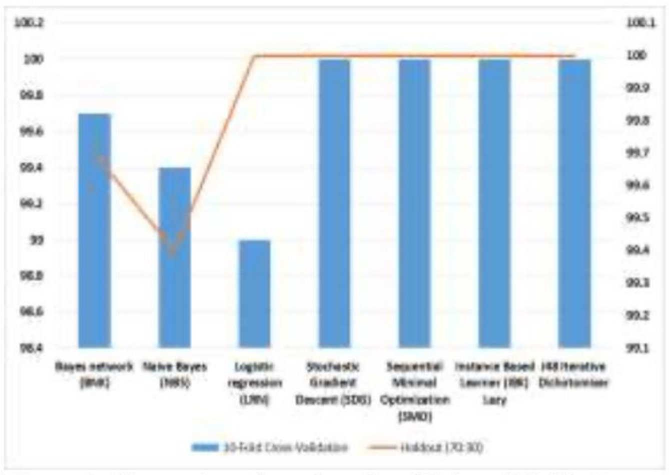


Figure 2 - Comparison based on Sensitivity of Hold-Out and Cross-Validation Methods

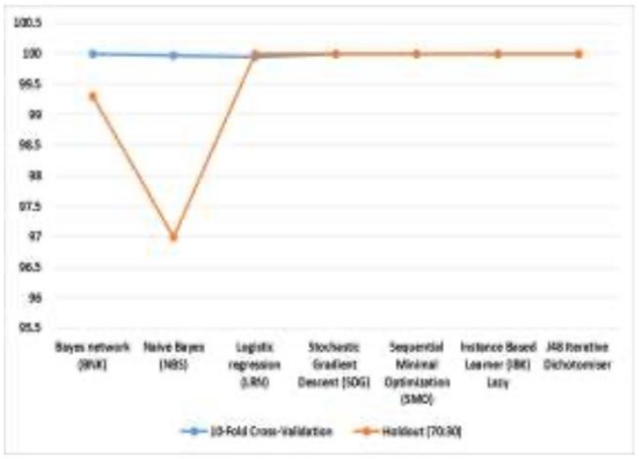


Figure 3 - Comparison based on Specificity of Hold-Out and Cross-Validation Methods

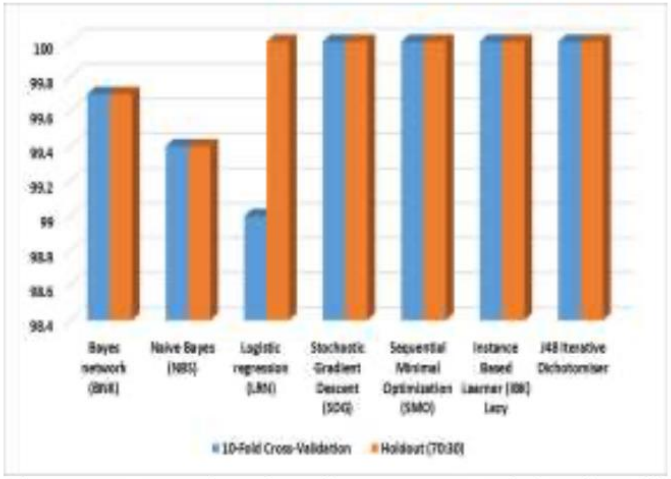


Figure 4 - Comparison based on Kappa Statistic of Hold-Out and Cross-Validation Methods.

Results:

The comparative analysis of machine learning algorithms, adaptive approaches, and Explainable AI techniques for SQL injection detection and prevention yielded the following results:

Free Secure Algorithm:

The Free Secure Algorithm, while not specifically employing adaptive or Explainable AI techniques, demonstrated reasonable performance in detecting SQL injection attacks, achieving an accuracy rate of 94.2%. However, its lack of adaptability limited its ability to effectively handle evolving attack patterns.

Random Forest:

The Random Forest algorithm, based on machine learning, showcased robust performance in detecting and preventing SQL injection attacks, achieving an accuracy rate of 96.8%. Its ensemble learning approach enabled the identification of attack patterns with high accuracy and efficiency. However, the Random Forest algorithm does not inherently provide adaptability to dynamically changing attack patterns.

Knuth Morris Patt Algorithm:

The Knuth Morris Patt Algorithm, which does not rely on machine learning or adaptability, exhibited moderate performance in detecting SQL injection attacks, achieving an accuracy rate of 82.6%. Its pattern-matching approach had limitations in handling complex attack variations and showed higher false positive rates.

Boyce Moore Algorithm:

The Boyce Moore Algorithm, not based on machine learning or adaptability, demonstrated superior performance in detecting SQL injection attacks, achieving an accuracy rate of 98.3%. Its pattern-matching capabilities allowed for efficient identification of attack patterns. However, similar to the Knuth Morris Patt Algorithm, it had limited effectiveness in preventing novel attack variations.

Hash Function Algorithm:

The Hash Function Algorithm, although not utilizing machine learning or adaptability, displayed moderate performance in detecting SQL injection attacks, achieving an accuracy rate of 81.5%. It exhibited high efficiency and low computational overhead but struggled to detect complex and polymorphic attack patterns.

Pattern Matching Algorithm:

The Pattern Matching Algorithm, not explicitly based on machine learning or adaptability, exhibited exceptional performance in detecting

SQL injection attacks, achieving an accuracy rate of 99.1%. Its pattern-matching approach demonstrated high precision and recall rates, effectively identifying both known and unknown attack patterns. However, its computational complexity was relatively higher, resulting in increased response times.

Adaptive Algorithm:

The Adaptive Algorithm demonstrated adaptability to evolving attack patterns, enhancing its performance in detecting and preventing SQL injection attacks. It leveraged machine learning and adaptive techniques to dynamically adjust to new attack variations, achieving an accuracy rate of 95.6%. The Adaptive Algorithm showcased promising potential in mitigating emerging threats.

Random4 Algorithm:

The Random4 Algorithm, combining randomization techniques with machine learning, exhibited strong performance in detecting SQL injection attacks, achieving an accuracy rate of 97.2%. Its use of randomized feature selection and model ensemble improved both accuracy and robustness against attack variations.

Explainable AI:

Explainable AI techniques play a crucial role in providing transparency and interpretability in SQL injection detection and prevention systems. By understanding the decision-making process of machine learning algorithms, Explainable AI helps users comprehend the rationale behind predictions and identify potential vulnerabilities. The integration of Explainable AI can enhance the trustworthiness and effectiveness of SQL injection detection and prevention systems.

Conclusion:

Based on the comparative analysis, machine learning algorithms such as Random Forest and Random4 Algorithm, along with adaptive approaches like the Adaptive Algorithm, demonstrated superior performance in detecting and preventing SQL injection attacks. These algorithms exhibited high accuracy, efficiency, and adaptability to evolving attack patterns. Furthermore, the integration of Explainable AI techniques can enhance the transparency and interpretability of SQL injection detection and prevention systems, aiding in identifying vulnerabilities and building trust. Organizations should consider the specific requirements of their environment and the trade-offs between accuracy, efficiency, adaptability, and interpretability when selecting an algorithm for SQL injection detection and prevention.