The ISC Int'l Journal of Information Security

January 2024, Volume 16, Number 1 (pp. 1–16)
http://www.isecure-journal.org

Modified Parse-Tree Based Pattern Extraction Approach for Detecting SQLIA Using Neural Network Model

Meharaj Begum A [1,*] and Michael Arock [1]

[1] *Bio-informatics and Parallel Computing Lab,Department of Computer Applications, National Institute of Technology, Tiruchirappalli, Tamilnadu, India*

ARTICLE INFO.

Article history:
Received: November 18, 2022
Revised: March 21, 2023
Accepted: April 17, 2023
Published Online: September 30, 2023

Keywords:
Modified Parse Tree, Neural Network Model, Pattern Mining, Siblings, SQLIA Detection, Web Application Security

Type: Research Article

doi: 10.22042/isecure.2023.370697.886

dor: 20.1001.1.20082045.2024.16.1.1.5

ABSTRACT

Whatever malware protection is upcoming, still the data are prone to cyber-attacks. The most threatening Structured Query Language Injection Attack (SQLIA) happens at the database layer of web applications leading to unlimited and unauthorized access to confidential information through malicious code injection. Since feature extraction accuracy significantly influences detection results, extracting the features of a query that predominantly contributes to SQL Injection (SQLI) is the most challenging task for the researchers. So, the proposed work primarily focuses on that using modified parse-tree representation. Some existing techniques used graph representation to identify characteristics of the query based on a predefined fixed list of SQL keywords. As the complete graph representation requires high time complexity for traversals due to the unnecessary links, a modified parse tree of tokens is proposed here with restricted links between operators (internal nodes) and operands (leaf nodes) of the WHERE clause. Tree siblings from the leaf nodes comprise the WHERE clause operands, where the attackers try to manipulate the conditions to be true for all the cases. A novelty of this work is identifying patterns of legitimate and injected queries from the proposed modified parse tree and applying a pattern-based neural network (NN) model for detecting attacks. The proposed approach is applied in various machine learning (ML) models and a neural network model, Multi-Layer Perceptron (MLP). With the scrupulously extracted patterns and their importance (weights) in legitimate and injected queries, the MLP model provides better results in terms of accuracy (97.85%), precision (93.8%), F1-Score (96%), and AUC (97.8%).

© 2024 ISC. All rights reserved.

1 Introduction

As per the IBM X-Force analysis of IBM Managed Security Services (MSS) data, SQLIA is one of the most powerful and prominent attacks (Michelle *et al.*, 2017)[1]. Two-thirds of web application attacks are caused by SQLI. It continues to be an effortless way for cyber criminals to do data breaching in a database. Although it is the oldest one, still a critical attack vector due to its strength. Many prominent data breaches have been the result of SQLI. In New York, the e-commerce website has been hacked and credit card information was breached in May 2020.

* Corresponding author.

Email addresses: mehartrichy@gmail.com, michael@nitt.edu

ISSN: 2008-2045 © 2024 ISC. All rights reserved.

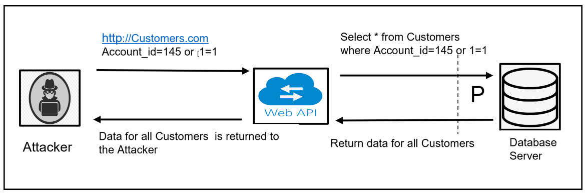

http://Customers.com
Account_id=145 or 1=1
Select * from Customers
where Account_id=145 or 1=1
Web API
P
Attacker
Data for all Customers is returned to the Attacker
Return data for all Customers
Database Server

Figure 1. SQLIA in web applications

Frederik Company's more than 8 million users' mail IDs and password hashes were hacked in August 2020 (Pallavi *et al.*, 2020)[2]. The social media platform, GAB said that the site had been attacked by cyber criminals and more than 70 GB of data and 40 million posts were leaked by hacktivists in February 2021 (DouOlenick *et al.*, 2020)[3]. All these incidents are the result of SQLIA against their websites.

Figure 1 shows how SQLIA is performed in web applications. The payload which is sent from the client passes through the web server, and application server and finally reaches the database management system of the database server. That payload will be substituted in the dynamic SQL statement of the application server and executed in the database server. During execution, the query processor parses and executes it. This proposed classification model should be integrated into the application server's code before the SQL statement is transmitted to the database server for execution to identify and prevent injection. This model secures the data from injection even if the application has some vulnerabilities. In Figure 1, line P denotes our proposed model which could be plugged into the application server before the query execution in the database server.

Detecting these injected queries before it is being executed in the database layer is a challenge to the researchers. Although many researchers have focused on SQLI and implemented many solutions to fix it, still it's a significant threat to web users. Attackers are also constantly searching for SQLI vulnerabilities on the Internet. A web application is said to be SQLI vulnerable if it could allow an attacker to insert their SQL statement through the input field of the user interface. The root causes of SQLI vulnerabilities are trusting input without validation, using the same legacy techniques, and using data and code in the same SQL statement while executing them. This vulnerability affects web applications that use SQL- based databases such as MySQL, Oracle, and SQL servers. Attackers exploit these vulnerabilities to intrude without any authentication into the database which contains confidential data and could also obliterate the database. There are three major categories of SQLIs, In-band SQLI (Error-based SQLI and Union-based SQLI), Inferential or Blind SQLI (Boolean-based and Time-based SQLI), and Out-of-band SQLI(Tautology-based).

Commonly used mechanisms to defend against attacks are validation functions, web application firewalls (WAFs), and prepared statements. But these models are not sufficient to protect the data. To overcome this problem, SQLI should be handled in the code before it is executed in the database server. Many ML models were already developed to detect injected queries. But the existing approaches used a pre-defined list of SQLIA signature keywords, operators, and symbols for identifying injected queries. If any symbol or keyword is ignored, it will not predict the attack perfectly. So, it has been found that the current techniques are not completely successful to detect SQLI. Hence, we require considering the general signature of legitimate and injected queries. The main target of this work is to obtain the signature (pattern) of the query using tree representation.

In the model, SQLiGOT, (Kar *et al.*, 2016)[4] proposed the usage of a complete graph for representing all the tokens of the WHERE clause with the assignment of weight between every pair of tokens based on the distance using a sliding window. The graph representation had many unnecessary connectivities between every pair of tokens, even though the relationship between the tokens exists only within its expression. The above paper is the motivation behind this proposed work to represent the query as a modified parse tree and train the model with scrupulously chosen features from the tree. This paper proposes an approach that overcomes these limitations based on the following five contributions:

(1) Using tree representation to maintain connectivity between only the relevant operators and operands in the WHERE clause expressions of SQL query.
(2) Implementing a single modified parse tree to find the patterns more effectively.
(3) Normalizing the nodes of the tree using a lookup table of keywords and symbols to maintain generality.
(4) Analyzing the patterns in siblings by giving weightage using conditional probability along with label encoding.
(5) Training MLP neural network model to recognize patterns and classify them effectively

The remaining work is organized as follows:

Research findings and literature-related issues are presented in Section 2. Section 3 describes the architecture of the proposed model, whose primary components are defined by the algorithms. Section 4 presents the results of an experimental evaluation along with dataset construction, hyperparameter optimization, time complexity analysis, results, and analysis. The paper concludes with a note about the future research objectives in Section 5.

2 Related Work

(Sajjadi *et al.*, 2013; Som *et al.*, 2016; Lawal *et al.*, 2016; Nagpal *et al.*, 2017; Alwan *et al.*, 2017; Ogheneovo *et al.*, 2013) [5–10]provided a detailed survey of types of vulnerabilities, types of SQLIA, and various existing techniques for identifying and preventing injections already discussed by researchers. (Jhala *et al.*, 2017)[11] explored advanced SQLIA named tautologically injected SQL queries and how they affected the data repositories. Existing research approaches dealt with various approaches such as ontology-based approach, pattern-based approach, ML, and NN models.

2.1 Ontology-Based Approaches

(SahaRoy *et al.*, 2014)[12] discussed how to implement POS Tagging and statistical word association scores (WAS) in ordinary web search queries. (Fang *et al.*, 2018)[13] proposed a model for classifying benign and malicious queries in which word-to-vector(W2V) and bag-of-words(BOW) concepts were used to find the vector of each query. Finally, they have concluded that the W2V attained higher accuracy than the BOW feature. (Abaimov *et al.*, 2019)[14] created a separate text file like a dictionary for providing encoding rules in which the list of operators, symbols, and keywords are numbered. Based on this file content, encoding will be done. If any symbol is missed, it will make the wrong prediction. (Jemal *et al.*, 2019)[15] have presented an Ontology-based NN model for classifying queries, which achieved 83.33% accuracy. All these researches were based on domain dictionaries with a fixed list of SQL keywords. So, they won't detect the SQLIA with dynamic structure.

2.2 Pattern-Based Approaches

(Rawat *et al.*, 2012)[16],(Latchoumi *et al.*, 2020) [17]incorporated the SVM model for SQL Injection Attack Detection by training it with SQLI signature keywords and applying string-matching techniques to classify them. (Uwagbole *et al.*, 2017)[18] used Two-Class Linear Regression and Two-Class SVM to create a classification system. The RegEx pattern is used to generate and train member strings. But its time complexity is high. They trained a dataset considering patterns. (Kranthikumar *et al.*, 2020)[19] have proposed a REGEX classifier that uses regular expressions as a filter to classify the applied query SQLI or genuine query. They used only eleven REGEX patterns of injected queries. They compared SVM, Gradient Boosting algorithm, and Naive Based classifier with REGEX classifier for detecting SQLIAs. This model produces 97% accuracy. (Hadabi *et al.*, 2022)[20] have proposed a model which is using a proxy in the middle between the client and server to check any input that is forwarded to the database server and prevent the attack using patterns that are implemented using regular expression. In this experiment, they use the XAMPP as a web server and Web Scrap proxy from OWASP with the virtualized environment as a proxy server. REGEX defines certain kinds of patterns only. So they are not sufficient to detect the injection.

2.3 Machine Learning Techniques

(Joshi *et al.*, 2014)[21] used a Nave Bayes ML technique to create a traditional model by counting the number of times each term appeared in a SQL query as a feature. This model gave 93.3% accuracy. (Mamun *et al.*, 2016)[22] discussed the K-Nearest Neighbors algorithm and a pattern recognition method for finding malicious URLs using lexical features of URLs with assigned weight by using the Euclidean distance function. (Kar *et al.*, 2016)[4] implemented a model of creating a graph of tokens for which they initially substituted each element in a query with a predefined name. Tokens are considered as vertices and their interaction is considered as edges. They assigned weights on edges based on the window size and closeness between the nodes. They calculated the degree of each node and used it as a feature for training the model in SVM which obtained 96.23% accuracy and a 4% false-positive rate. (Yu *et al.*, 2019)[23] have proposed an SVM model in which the query is transformed into an Eigenvector for identifying SQLI. (Arumugam *et al.*, 2019)[24] have extracted the flow variables such as the IP address of the source, date and time of the request, payload sent from the requesting app, and target application from the .csv file, and signature keywords of SQLI are extracted from the request parameter. They applied a Logistic Regression classifier for detection with 70% accuracy only. (Li *et al.*, 2019)[25] have proposed an adaptive deep forest model for SQLI detection in which thirty features are considered for classification. They compared KNN, SVM, RF, and Naive Bayes with Adaptive Deep Forest models (ADF) and found out ADF is better than all other models (Triloka *et al.*, 2022)[26] have proposed a comparative study of various machine learning models to detect SQLIA and suggested na¨ıve Bayes classifier with the accuracy of 97.5%.

2.4 Neural Network Models

(Sheykhkanloo *et al.*, 2017)[27] formed a 32-bit vector for signatures of each query and an 8-bit vector for SQLI types. These vectors are calculated for each query and represented as columns of the input matrix and target matrix respectively. They used the NN model for classification. (Tang *et al.*, 2020)[28] proposed a unique model combining MLP and LSTM with extracted characteristics from URLs such as URL length, number of keywords, keyword weight, number of special characters, and so on. Recognizing keywords and characters other than the given list is extremely poor in this model and it faces an overfitting problem. (Zhang *et al.*, 2022)[29] proposed the SQLNN model uses the TF-IDF of tokens of SQL query as input features with an accuracy of 96.16%. (Falor *et al.*, 2022)[30] proposed a CNN model to detect SQLIA which first breaks up the cluttered text documents containing the payload queries into separate queries based on their type and classify them using convolutional neural network accordingly.

3 Proposed Approach

SQLI is a cyber-security risk that can damage web applications and databases. With SQLI, hackers can access databases and networks without authentication. They inject malicious script into web form input fields, which then alters the WHERE clause conditions and results of the query.

A vast array of ML and NN models have been proposed for detecting SQLI, by extracting features mostly based on characteristics of SQL queries such as length, characters, spaces, keywords, symbols, etc., As the feature extraction accuracy significantly impacts the detection results, these features were not sufficient to detect all the complex SQLIAs and their discussions did not cover generalized patterns (signature) for finding SQLIAs. So, pattern-based feature extraction is proposed here. Initially, the query is transformed into the tree structure, then its' pattern is extracted from that structure. These patterns are trained in a NN model to detect SQLIAs.

Figure 2 shows the architecture of the proposed model. In this figure, Q denotes the input query, and the list of segments L_0, L_1, and L_2 represent the 'SELECT' clause, 'FROM' clause, and WHERE clause segments of parsed SQL query respectively. The set N=N_1, N_2, N_3,... represents the nodes set generated from the tokens of the WHERE clause, and the set E= E_1, E_2, E_3,....E_n represents the edge set generated from the pairs of tokens e.g. E_1=(N_1, N_2), E_2=(N_1, N_3), etc., where E_1 is an edge between N_1 and N_2, E_2 is an edge between N_1 and N_3. T is a Modified parse Tree constructed in a specific manner, POL is a post-order traversal of tree T consisting of a list of tokens, the set S=(S_1, S_2), (S_3, S_4).... is a set of siblings pair obtained from POL, the set W= W_1, W_2, W_3,.... is a probabilistic weight vector for

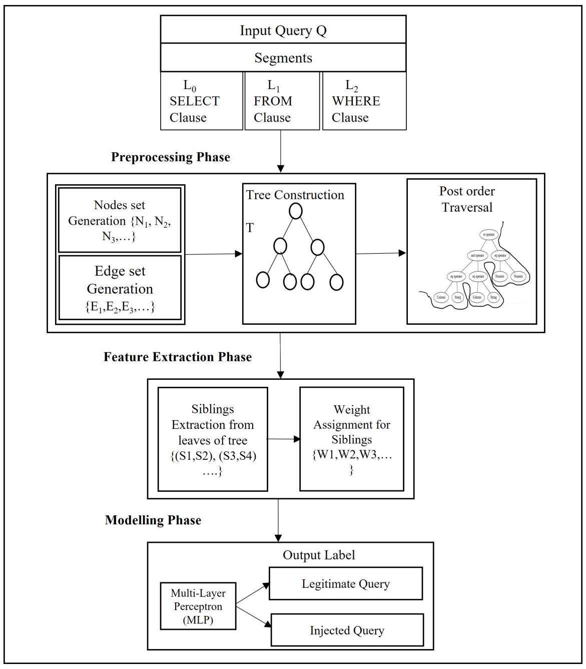

Input Query Q
Segments
L_0 SELECT Clause
L_1 FROM Clause
L_2 WHERE Clause
Preprocessing Phase
Nodes set Generation {N_1, N_2, N_3,…}
Edge set Generation {E_1,E_2,E_3,…}
Tree Construction
T
Post order Traversal
Feature Extraction Phase
Siblings Extraction from leaves of tree {(S1,S2), (S3,S4) ….}
Weight Assignment for Siblings {W1,W2,W3,… }
Modelling Phase
Output Label
Multi-Layer Perceptron (MLP)
Legitimate Query
Injected Query

Figure 2. Architecture of proposed model

extracted siblings, MLP is a Multilayer perceptron model which takes the weight vector as input and classifies the given query Q as a legitimate query or injected query.

The proposed work specifically deals with the WHERE clause of the SQL query which is transformed into a tree of tokens, and patterns are extracted from sibling nodes of the tree by traversing the tree in post order. Initially, the query is processed with the built-in parser 'Moz SQL parser' and generates a list of sub-lists of tokens for the SELECT, FROM, and WHERE clause segments. As the proposed work focuses just on the WHERE clause, tokens related to it alone are taken from the parsed list. Each token represents a node, and edges are formed from the tokens in a specific way, as described in the CONSTRUCT TREE () algorithm, resulting in a modified parse tree. The WHERE clause is a collection of logical expressions and relational expressions. Each expression in the WHERE clause is represented as a subtree. Then, all the sub-trees are connected to form the parse tree for the entire WHERE clause. The leaf nodes of the tree contain the pattern of each expression. The siblings' pair from the leaf nodes for each parent form the patterns of the query. They are the primary feature of the input query. Legitimate and injected queries differ in their siblings' patterns and their uniqueness helps classify the queries. The patterns of these siblings are considered as the features of that query and are given a weight based on the conditional probability. The weight vector is trained using MLP for the classification.

Pseudocode 1 SQL Injection Classifier

```
1: function SQL-INJ-CLASSIFIER(query)
   ▷ Built-in Parser is used
2:     $L \leftarrow$ Parse(query)
   ▷ List of tokens of WHERE clause is assigned to WLT
3:     $WLT \leftarrow$ List of tokens of $L_2$
   ▷ T is the Modified Parse Tree of WHERE clause
4:     $T \leftarrow$ Call CONSTRUCT-TREE(WLT)
   ▷ POS is the Post order traversal of normalized parse tree
5:     $POL \leftarrow$ Call POST˙ORDER˙TRAVERSAL(T)
   ▷ SL – List of Siblings Pair
6:     $SL \leftarrow$ Call GET˙SIBLINGS(POL)
   ▷ using formulae in Eq. 1 through 8
7:     Assign-weight for all siblings
8:     Train the Neural Network model to classify the query label as 'Legitimate' or 'Injected'
9:     Display Label
10: end function
```

A complete main algorithm SQL INJ CLASSIFIER () which comprises sub-algorithms such as CONSTRUCT TREE () and GET SIBLINGS () is presented here which gets the query as input and performs pre-processing steps such as parsing, segmentation, and tokenization. After that, the modified parse tree is generated to extract the features, then the extracted features are assigned weights. Finally, MLP is trained with suitably chosen features and produces the output as a legitimate query or injected query.

3.1 Parsing, Segmentation, and Tokenization

The first step of this work is to parse the input query using the built-in parser 'Moz sql parser', which produces the list of sub-lists L, for SELECT clause, FROM clause, and WHERE clause tokens. These segments are labeled as L_0, L_1, and L_2 respectively. The tokens from the WHERE clause segment, L_2, are extracted and a modified parse tree is constructed by sending these extracted tokens to the algorithm CONSTRUCT TREE ( ). For example, consider two sample queries Q_1, Q_2, and tokens associated with the WHERE clause of them are shown below,
Legitimate query Q1: Select title, phone, hire_date from employees where first_name = 'Nancy' and last name = 'Edwards'.
Injected query Q2: Select * from suppliers where sup_id =' SUP0145' and city='DELHI' or 1=1.
Tokens list of WHERE clause of Q1:('first_name', '=', 'Nancy', 'and','last_name', '=' , 'Edwards').
Tokens list of WHERE clause of Q2: ('supp_id, "=", 'SUP0145', 'and', 'city', "=", 'DELHI', 'or', 1, "=", 1)

3.2 Constructing a modified parse tree

Pseudocode 2 CONSTRUCT-TREE

```
1: function CONSTRUCT-TREE(WLT)
   ▷ Assign list of tokens, WLT to nodes list, N.
2:     $N \leftarrow WLT$
3:     By traversing through the list N, construct a sub tree for each logical expression such that the logical operator is the parent node and left, right-hand side relational expressions are its children.
4:     Then, expand each relational expression and construct a sub tree for each of them such that the relational operator is the parent node, and its left operand and right operand are its children.
5:     Link all the sub trees based on their order of operator precedence in the WHERE clause.
6:     The resultant parse tree contains either root or internal nodes as operators and leaf nodes as operands.
7:     Normalize the labels of nodes using a predefined list of tokens in SQL.
8:     The final tree contains all the nodes being normalized.
9:     return T
10: end function
```

The algorithm CONSTRUCT_TREE ( ) gets input from a list of WHERE clause tokens, denoted as nodes list, N= N_1, N_2, N_3,. . . N_n and generates the modified parse tree. The modified parse tree has some properties such that every logical expression of the WHERE clause is represented as a sub-tree in which the logical operator is the parent node, and left and right-hand side relational expressions are its children. Tree is constructed by defining edge set E= E_1,E_2,E_3,. . . .E_n. The edge $E_i = (N_i, N_j)$ can be formed from Nodes set N, such that the parent node N_i should be an operator and the child node N_j maybe of any category. It results in all the operators being internal nodes and operands being leaves of the tree. According to the order of operator precedence in the WHERE clause, all the sub-trees are linked to the root to form a single tree.

The labels of tree nodes should be normalized because generic tokens are required to find the pattern. Each node from the tree is mapped to the list of predefined SQL operators, keywords, special characters, and built-in SQL functions. Finally, the labels of tree nodes are normalized. Table 1 shows the sample list of predefined tokens and their normalized label substitution. Figure 3 describes how the query is transformed into a normalized parse tree.

Table 1. Sample list of predefined tokens and their normalized label substitution

Sl.No.	Tokens	Normalized Label Substitution
1.	Len, Current_User, Sin, Datefromparts, Pi, Sqrt, Power, Current_Timestamp Dateadd, Translate, Getdate, Getutcdate, Sysdatetime, Difference, Session_User, Convert, Datediff, Square, Day, Soundex, Ltrim, Replace, Datepart, Nullif, Coalesce, Trim, Nchar, Sessionproperty, Atan, Quotename, System_User, Upper, Ascii, Charindex, Format, Floor, Concat_Ws, Concat, Tan, Round, Cast, Lower, Reverse, Str, Rand, Isnull, Atn2, Datalength, Radians, Sign, Space, Acos, Isdate, Isnumeric, Year, Stuff, Datename, Exp, User_Name, Replicate, Substring, Max, Abs, Unicode, Iif, Avg Ceiling, Month, Count Cos, Sum, Log, Asin, Log10, Min, Degrees, Left, Cot, Right, Char	SQL Functions
2.	Left_Join, Not_Between, Left_Outer_Join , Not_Like, Order_By , As, Between , Join , Union, Limit, End , Inner_Join , Else , Union_All, Full_Outer_Join, Not_In , Cross_Join , Using , Is_Not, In, Then, With, , ight_Outer_Join, Desc , Full_Join , Reserved, Collate, Is, From , Offset, Right_Join, Asc, Where, Like ,Group_By, _Nocase, Having , Case, When	Keywords
3.	∼,-, not	Operators
	+,-,*,/,mod,=,<>,not between, or , and	
4.	*, !, @, #, $, %, ∧, &, *, _, -, *,!,@,#,$, %,∧,&, *,_, -	Special Characters
5.	Select	Sub query

3.3 Post-Order Traversal

The tree nodes are visited in post-order to get the node's list in a specific order to discover the siblings from the leaf nodes. The operators should be excluded from the traversal list because the proposed work is based on the siblings (operands) to construct patterns since most of the hackers inject malicious data here. In the post-order traversal, the sequence of operands is given first, followed by their appropriate operators. Finally, it is easy to read backward from the end of the post-order list and exclude the element at last. So, post-order traversal is a better option for locating siblings than pre-order and in-order traversals. Figure 4 shows the post-order traversal of query Q_2.

Post-order Traversal for Q_1: ['Column', 'String', 'eq operator', 'Column', 'String', 'eq operator', 'and operator']

Post-order Traversal for Q_2: ['Column', 'String', 'eq operator', 'Column', 'String', 'eq operator', 'and operator', 'Numeric', 'Numeric', 'eq operator', 'or operator']

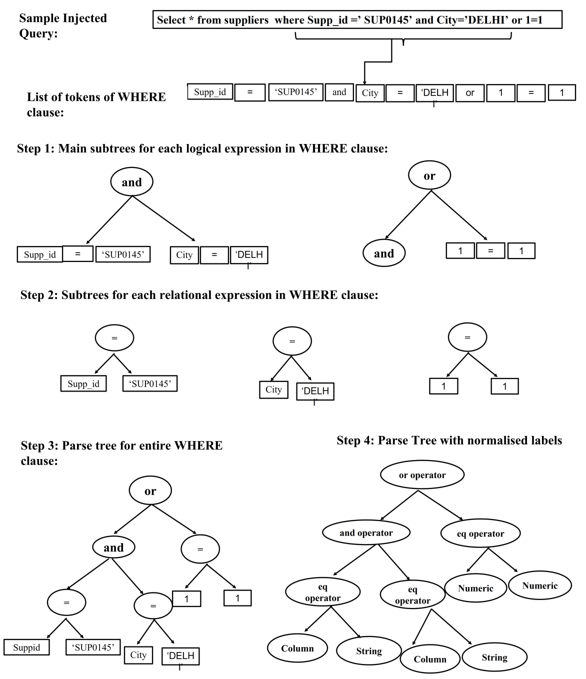

Sample Injected Query:
Select * from suppliers where Supp_id =' SUP0145' and City='DELHI' or 1=1
List of tokens of WHERE clause:
Supp_id = 'SUP0145' and City = 'DELH or 1 = 1
Step 1: Main subtrees for each logical expression in WHERE clause:
and
Supp_id = 'SUP0145' City = 'DELH
or
and
1 = 1
Step 2: Subtrees for each relational expression in WHERE clause:
=
Supp_id 'SUP0145'
=
City 'DELH
=
1 1
Step 3: Parse tree for entire WHERE clause:
or
and
=
=
=
1 1
Suppid 'SUP0145' City 'DELH
Step 4: Parse Tree with normalised labels
or operator
and operator
eq operator
eq operator
eq operator
Numeric
Numeric
Column String Column String

Figure 3. Construction of parse tree

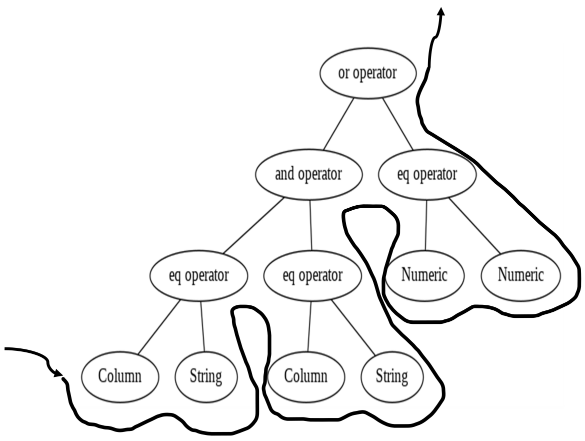


Figure 4. Post order for Q_2

3.4 Finding Siblings

In a modified parse tree, the parent nodes are always operators, branches are operands, and the leaf nodes are operands of the entire WHERE clause. Since the proposed work is concerned with the patterns of the operands of WHERE clause expressions, only the leaf nodes are considered and passed to the algorithm GET SIBLINGS (). While considering the degree of each node, leaf nodes have a degree of zero as they

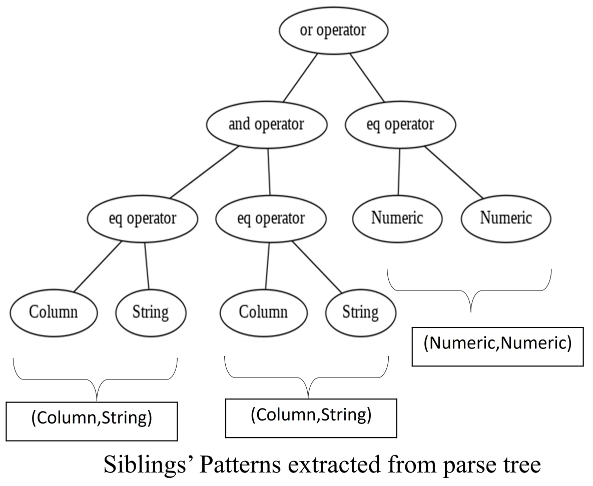


Figure 5. Siblings for query Q2

have no descendants. So, nodes with zero degrees are considered as leaves, and others are discarded from the list. Then the leaves that have the same parent are thus referred to be siblings. Figure 5 shows the siblings of the sample query Q2.
Siblings for Q1: [['Column', 'String'], ['Column', 'String']]
Siblings for Q2: [['Column', 'String'], ['Column', 'String'], ['Numeric', 'Numeric']]

Pseudocode 3 Algorithm GET-SIBLINGS(POL)

```
1: function GET-SIBLINGS(POL)
2:     Find out the leaf nodes by finding the nodes with degree 0.
3:     Leaf nodes ← $\forall N_i \in N$ from $POL$ where $deg(N_i) = 0$
4:     Find out the list of siblings' pair, SL, by combining the nodes with the same parent.
5:     SL ← $(N_i, N_j) : N_i$ and $N_j \in$ Leaf nodes and $Parent(N_i) = Parent(N_j)$
6:     return list of siblings' pair, SL
7: end function
```

3.5 Assigning Weight

Each expression in the WHERE clause is represented as a siblings' pair now. Each pair is a pattern. The list of extracted siblings' pairs for each query defines the pattern of the query. Uniqueness in this patterns list helps us classify the query. This pattern will become the main feature for training the model. Each query pattern will be assigned a weight based on its probabilistic score using conditional probability and Bayes theorem.

3.5.1 Set Creation

The extracted sibling's pairs for the entire dataset are segregated into two sets based on its label, one contains legitimate pairs and another one contains injected pairs. The intersection of these two sets contains the pairs common in both sets.
Sample sets:
Legitimate Pairs set L = Column- String, Column-Numeric, Column- SQL functions, Column- Sub-query, etc.,
Injected Pairs set I = String- String, Numeric-Numeric, Column- String, Column-Numeric, Column-SQL functions, SQL functions-SQL functions, Special chars- Special chars, Empty string- Empty string, Empty string- Numeric, Special chars-Empty string, etc.,
Intersection Set$L$**∩I**= Column- String, Column-Numeric, Column- SQL functions, etc.,

3.5.2 Encoding with a Weight Assignment

Label encoding and conditional probability are combined and applied to the pairs to give weightage to them. The dataset contains queries of two class labels legitimate and injected. Initially, the label encoding technique is applied for all the sibling pairs extracted from the dataset. Each pair has assigned one unique encoded value. But this is not adequate for the weight assignment since a siblings' pair may occur in both sets (Legitimate and Injected). Its score should reflect its contribution in its category (Legitimate or Injected). So, conditional probability is combined to differentiate them based on their contribution.

It requires the probability of the legitimate sibling's pairs and malicious sibling's pairs. It is appropriate to apply the Bayes theorem in this case. Because it describes the probability of occurrence of an event related to any prior condition. In this case, the probability of occurrence of a sibling's pair depending on the condition of whether it belongs to a legitimate query or injected query must be calculated. It is known as conditional probability. As a result, Bayes's theorem is applied here to calculate the weight.

According to the Bayesian theorem, the probability of an event can be determined by prior knowledge of the potential conditions that could influence it. By knowing the conditional probability (shown in Eq.(1)), we can use the Bayes rule to determine the reverse probability.

$$P\,(\mathrm{A} \mid \mathrm{B}) \quad = \frac{\mathrm{P(B|A).P(A)}}{\mathrm{P(B)}} \tag{1}$$

Where

(1) The probabilities of events A and B are denoted

by P(A) and P(B), respectively.
(2) The probability of A given B is P(A|B).
(3) The probability of B given A is P(B|A).

The contribution of the siblings' pattern in a particular label can be calculated as shown in Eq.(2)

$$P\left(SiblingsPattern \mid Label\right) = \frac{P\left(Label \mid SiblingsPattern\right) * P\left(SiblingsPattern\right)}{P\left(Label\right)} \tag{2}$$

The probability of a specific Siblings' pattern is P(Siblings' pattern).

(1) P(Label) is the probability of the Label (Legitimate or Injected).
(2) P(Siblings$'$ pattern|Label) is the probability of the particular siblings' pattern given the Label (Legitimate or Injected).
(3) P(Label|Siblings$'$ *pattern*) is the probability of the Label either Legitimate or Injected having the Siblings' pattern.

$$P\left(\left('\mathrm{Column}','\mathrm{Str}'\right) \middle| \mathrm{Legitimate}\right) = \frac{\mathrm{P(Legitimate} \mid \mathrm{(Column, Str)) .P(\,(Column, Str)\,)}}{\mathrm{P\,(Legitimate)}} \tag{3}$$

$$P\left(\left('\mathrm{Column}','\mathrm{Str}'\right) \middle| \mathrm{Injected}\right) = \frac{\mathrm{P(Injected} \mid \mathrm{(Column, Str)) .P(\,(Column, Str)\,)}}{\mathrm{P\,(Injected)}} \tag{4}$$

Based on the generated probabilistic scores, the corresponding encoded value is changed. Thus, the label sequence for the entire query is generated. The sequence is fed into the MLP for training. The sequence having at least one injected label will be treated as injected, otherwise legitimate. The occurrences of certain siblings in certain sequences could improve the classification. This model can categorize any query input based on its label pattern.
The following three equations are used to give weights,

(1) Keep the encoded value as 0 if the sibling's pair SP is in the injected set alone.

$$Weight\ W[SP] = \ 0, \quad \text{where } SP \in I \tag{5}$$

(2) The encoded value should remain unchanged if the sibling's pair SP is in a legitimate set alone.

$$Weight\ W[SP] = LE[SP], \tag{6}$$

where $SP \in L$ and LE is label encoded value.
(3) When a sibling's pair is in the intersection set (in both sets), multiply the probabilistic score by its encoded value.The probabilistic score P(SP) can be calculated as Eq.(7) for Scaling up or down of corresponding SP contribution and substituting in Eq. (8) to get the weight value of SP.

$$P(SP) \ = P(SP|Legitimate) * P(SP|Injected) \tag{7}$$

$$Weight\ W[SP] = P(SP) \cdot LE[SP], \tag{8}$$

where $SP \in L \cap I$.

There are 22 unique sibling pairs extracted from the dataset. Their importance in the specific label is mentioned as the percentage with bold letters and their dominance in the label is shown in Table 2. Table 3 lists the sample queries along with their labels, siblings' pairs, and the set they belong to. Whenever the siblings are part of a Legitimate pair, the initial weight will be based on the encoded value of the pair. Injected siblings' pairs are assigned a weight of zero. When it belongs to both sets, its initial weight will be calculated by conditional probability as shown in Table 4.

To avoid a discontinuity in the range of values, the exponential average of W[SP] is calculated here. Table 5 demonstrates how the exponential average of W[SP] is used to get the final weight of the SP. It offers 1 for injected pair, maximizes the value for L∩I pair, and minimizes the value for the legitimate pair. Hence, there is no discontinuity in the range of values as shown in Figure 6. The suggested model would use weights with threshold values to categorize the labels.

When a sibling's pair is in the intersection set (in both sets), multiply the probabilistic score by its encoded value. The probabilistic score P(SP) can be calculated as Eq.(3) and substituted in Eq. (4) to get the weight value of SP.

$$P(\text{SQL functions, numeric} \mid \text{Legitimate}) = \frac{\frac{1}{3}\cdot\frac{3}{6}}{\frac{2}{6}} = \frac{1}{2} = 0.5 \quad \text{(From Equation 3)}$$

$$P\left(SQL\ functions, numeric\right)|Injected) = \frac{\frac{2}{3}\cdot\frac{3}{6}}{\frac{4}{6}} = \frac{2}{4} = 0.5 \quad \text{(From Equation 4)}$$

$$P\left(SQL\ functions, numeric\right) = 0.5 * 0.5 = 0.25 \quad \text{(From Equation 7)}$$

$$W\left(SQL\ functions, numeric\right) = 0.25 * 1 = 0.25 \quad \text{(From Equation 8)}$$

When the query has more than one siblings pairs, then their average will be taken for calculating the exponential average.

4 Experiments, Results, and Analysis

4.1 Dataset Construction

The dataset utilized in the proposed model is a collection of instances from Github (Yu *et al.*, 2018)[31] with legitimate and injected queries (records). Each query is assigned one of two labels: legitimate or injected. There are 3000 queries for each class label.

Table 2. List of siblings' patterns and their contribution to the query

Sl.No.	Siblings' Pattern	% of importance in Legitimate query	% of importance in injected query	The dominance of Pattern in Label
1.	(Column, String)	36.6	3.8	Legitimate
2.	(Column, Numeric)	32.2	3.2	Legitimate
3.	(Column, Empty string)	0	15.2	Injected
4.	(Column, Special chars)	0	2.6	Injected
5.	(Column, SQL functions)	4.8	1	Legitimate
6.	(Column, Sub-query)	12	4.6	Legitimate
7.	(Column, Minus Operator)	0	3.6	Injected
8.	(String, String)	0	9	Injected
9.	(Numeric, Numeric)	0	8.8	Injected
10.	(SQL functions, SQL functions)	0	6.6	Injected
11.	(Special chars, Special chars)	0	0.8	Injected
12	(Empty string, Empty string)	0	1.6	Injected
13.	(Empty string, Numeric)	0	0.8	Injected
14.	(Special chars, Empty string)	0	1.6	Injected
15.	(SQL functions, Numeric)	9	10.8	Injected
16.	(Numeric, SQL functions)	3.6	0	Legitimate
17.	(Minus Operator, Empty String)	0	6.8	Injected
18.	(String)	0	4.2	Injected
19.	(Numeric)	0	4.8	Injected
20.	(Empty string)	0	2.6	Injected
21.	(Special chars)	0	5.,8	Injected
22.	(SQL functions)	0	6.8	Injected

4.2 Experimental Setup

The dataset is trained using a variety of ML models, including SVM, Random Forest, KNN, Decision Tree, Naive Bayes, and a NN model, MLP. MLP outperforms other models in evaluation measures. Table 6 shows various models and their evaluation metrics:

Table 3. Sample queries

Sl. No	Input Query	Siblings	Label	Siblings Set
1	Select * from suppliers where Salary >50000 and Dept='Sales'	[(Column, numeric), (Column, String)]	Legitimate	[L, L]
2	Select * from suppliers where Salary >50000 and Round (Salary) =65000	[Column, Numeric), (SQL function, Numeric)	Legitimate	[L, $L\cap$I]
3	Select * from suppliers where 1=1	[(Numeric, Numeric)]	Injected	[I]
4	Select * from suppliers where Ucase(Dept)= 'SALES' and Round (Salary) =65000	[(SQL function, SQL function), (SQL function, numeric)]	Injected	[$L\cap$I, $L\cap$I]
5	Select * from suppliers where "AAA" ="AAA" and Ascii('A') =65	[(String. String), SQL functions, numeric)]	Injected	[I, $L\cap$I]
6	Select * from suppliers where "AAA" ="AAA" and 1=1	[(String. String), (Numeric, Numeric)]	Injected	[I, I]

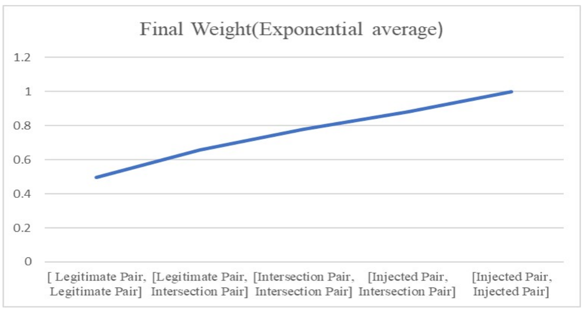


Figure 6. Siblings' pairs and their final weight

Table 4. Initial weight calculation

Set	Siblings' Pair	Label encoded value LE(SP)	Normalised LE(SP)	Initial weight W(SP)
$SP \in$I	(String, String)	1	1/5=0.2	0
$SP \in$I	(Numeric, Numeric)	2	2/5=0.4	0
$SP \in$L	(Column, String)	3	3/5=0.6	0.6
$SP \in$L	(Column, umeric)	4	4/5=0.8	0.8
$SP \in L \cap$I	(SQL functions, numeric)	5	5/5=1	0.25

Table 5. Final weight calculation for possible siblings pair

Case	Siblings pair	W[SP]	Average	Final Weight (Exponential average)
1.	[Legitimate Pair, Legitimate Pair]	[0.6,0.8]	[0.7]	e-0.7=0.496
2.	[Legitimate Pair, Intersection Pair]	[0.6,0.25]	[0.42]	e-0.42=0.6570
3.	[Intersection Pair, Intersection Pair]	[0.25,0.25]	[0.25]	e-0.25=0. 7788
4.	[Injected Pair, Intersection Pair]	[0,0.25]	[0.125]	e-0.125=0.8824
5.	[Injected Pair, Injected Pair]	[0,0]	[0]	e-0=1

accuracy, precision, recall, F1 score, and ROC (%).

4.3 Hyperparameters for MLP

The proposed NN model(MLP) is trained and tested using various hyperparameters for improved performance as shown in Table 7. The batch size determines how many samples will be processed before the internal model parameters are updated. At the end of the batch, the predicted output variables are checked with the actual values, and the difference is calculated. The update algorithm is used to improve the model based on this difference. Depending on the number of epochs, the learning algorithm will traverse the entire training set multiple times. After each epoch, the parameters of the internal model have been updated for all samples in the training dataset. There are one or more batches in an epoch.

Table 6. Models and their evaluation metrics

Model	Accuracy Score	Precision	Recall	F1 Score	ROC AUC:
SVM	0.926	0.926	0.926	0.926	0.926
Random Forest	0.956	0.932	1.0	0.965	0.946
KNN	0.964	0.962	0.961	0.982	0.975
Decision Tree	0.957	0.941	0.99	0.96	0.975
Na¨ıve Bayes	0.946	0.932	1.0	0.956	0.946
MLP (Proposed)	0.978	0.9383	0.992	0.964	0.978

Table 7. Hyperparameters for the proposed model

Sl.No.	Parameters	Value
1.	Batch size	32
2.	Epochs	10
3.	Optimizer	Adam
4.	Loss Function	Binary Cross-Entropy
5.	Learning Rate α	0.001
6.	Activation function	Output layer- Sigmoid Other layers-ReLu

The model is trained using a variety of optimizer and loss function combinations, including 'Adam and BCE', 'Adam and Hinge', 'RMSprop and BCE', and 'RMSprop and Hinge'. Owing to the binary nature of this problem, both the Adam optimizer and the Binary Cross-Entropy loss function deliver excellent performance at epoch 10 and provide high accuracy and precision.

Adaptive moment estimation or Adam is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm. It acts upon,

(1) The gradient component by using V, the exponential moving average of gradients (as in momentum), and
(2) The learning rate component is by dividing the learning rate α by the square root of S, the exponential moving average of squared gradients (as in RMSP).

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t + \epsilon}}.\hat{V}\mathrm{t}$$

Where,

$$\hat{V}\mathrm{t} = \frac{V_t}{1-\beta_1^t}$$

$$\hat{S}_t = \frac{S_t}{1-\beta_2^t}$$

are the bias corrections, and

$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2)\,[\frac{\partial L}{\partial w_t}]\ 2$$

with V and S initialized Proposed default values:

(1) α= 0.001 (learning rate or step size. The proportion that weights are updated).
(2) $\beta_1 = 0.9$ (The exponential decay rate for the first moment estimates).
(3) $\beta_2 = 0.999$ (The exponential decay rate for the second-moment estimates).
(4) ϵ= 10^{-9} (a small positive constant to avoid the 'division by 0' error when $V_t = 0$).

Loss Function: Binary Cross-Entropy/Log Loss

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{n} y_i.log\left(p\left(y_i\right)\right) + \left(1-y_i\right).log(1-p\left(y_i\right))$$

Where,

(1) y_i represents the actual class of the ith sample.
(2) p(y_i) is the predicted probability that the ith sample belongs to class 1.
(3) 1-p(y_i) is the corrected probability that the ith sample belongs to the actual class.

The negative average of corrected projected probabilities is known as Binary Cross Entropy. It compares each projected probability to the actual class outcome, which can be 0 or 1. The score is subsequently calculated, which penalizes the probabilities based on their deviation from the expected value. This relates to how similar or dissimilar the value is to the real one. At Epoch 10 and Batch Size 32, the model obtained better accuracy of 97.8%, precision of 93.83 %, recall of 99.27 %, F1 score of 96.47 %, and area under the ROC curve of 97.8 %.

The final layer employs the sigmoid activation function, which compresses all values between 0 and 1 into a sigmoid curve. The activation function for the remaining layers is ReLU (Rectified Linear Units). ReLU is a half-rectified function, which means that the value is 0 for all inputs less than 0, but the value is kept as it is for anything positive. There is just one output unit because a probability will be predicted for each record value in the input query pattern. The query is injected if it is above 0.9. It's not an injected query if it's smaller than 0.2.

4.4 Results and Analysis

The proposed model has experimented in the system with an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz 3.40 GHz CPU, 8 GB of RAM, 64-bit operating system, Win 10, and x64-based processor. Figure 7 shows a graphical representation of the metrics comparison between different combinations of optimizers and loss functions at epochs 1,5,10, and 15. The ROC curve depicts the ratio between the false positive rate and the true positive rate as 97.8%. According to the training and validation graph, the gap between training loss and validation loss is minimum in this suggested model (see Figure 8 and Figure 9.)

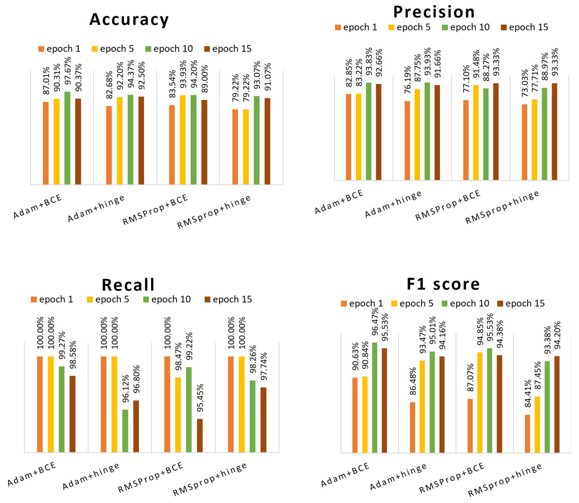


Figure 7. Comparison of metrics

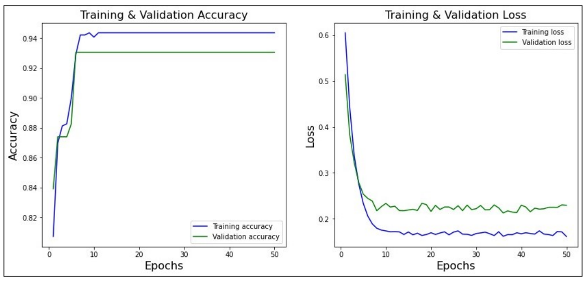


Figure 8. Accuracy and loss analysis

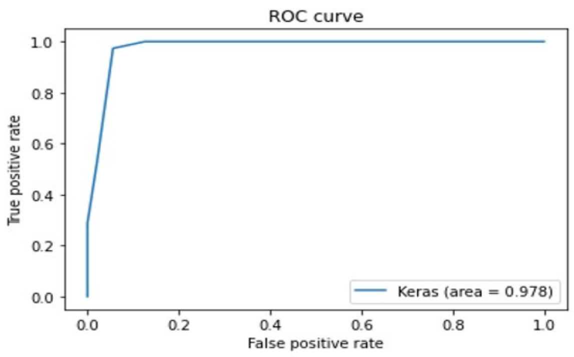


Figure 9. Roc curve

4.5 Scope of the Proposed Model

This proposed model has classified the query with multiple conditions, simple sub-query, and natural join accurately. Also, it can classify the views as its WHERE clause structure is the same as the table. But queries with the nested query and union are not classified exactly as shown in the 5th and 6th entries of Table 10.

The subqueries require special attention to computationally parse them. In normal queries, only the WHERE clause is prone to injection and its tree representation is mostly balanced but in subqueries as well as nested queries, the trees are skewed, and they must be analyzed with separate pattern extraction methodology instead of sibling's pair to find the injection within that. So, it will be extended in future work.

When the parser encounters the keyword "Select" in the WHERE clause, it is mapped as "Subquery". It is further manually parsed as shown in item 5,6 in Table 10, as recursive computational processing will generate skewed trees with multiple SELECT, FROM, and WHERE clauses. Also, compared to the proposed work, this scenario has a significantly different set of requirements for parsing, pruning, and mining. So, the dynamic structure of the nested query and union should be patterned properly and that will be considered as future work of this model. Table 10 shows some sample legitimate and injected queries and their classification results. This model is also capable of finding the injected query with a single malicious pair as well as a high number of benign pairs. Because the malicious pair has the highest weight for the injected query and it is taken for its classification.

4.6 Time Complexity

The modified parse tree construction has time complexity O(n log n) since the number of tokens in the WHERE clause is n. The post-order traversal of the tree has O(n) and the time complexity of finding its siblings' pattern is also O(n). So, the total time complexity is the sum of O(n log n), O(n), and O(n) which is equivalent to O(n log n). Table 11 shows the time complexity of the proposed work. When graphs are used as samples in ML projects, shortest paths or random walks are commonly employed and they have an O(n^3)time complexity and an efficient approach for computing betweenness and closeness centrality have a time complexity of O(nm + n^2logn) and O(n+m) respectively where n is the number of nodes and m is the number of edges as stated by (Kar *et al.*, 2016)[4] But, here the tree construction and post-order traversal have the maximum time complexity of O(n log n).

Table 8. Comparative analysis between existing and proposed models

Sl. No	Existing Model	Feature Considered	Techniques applied & Accuracy obtained
1.	Prediction of SQL Injection Attacks in Web Applications Arumugam, C., Dwarakanathan, V. B., (2019).	Source IP, Time, Target application	Logistic Regression 72%
2.	A Learning-based Neural Network Model for the Detection and Classification of SQL Injection Attacks Sheykhkanloo, N. M. (2017).	Pre-defined list of SQLi attack signatures	NN 90%
3.	SQL Injection Detection using Machine Learning Joshi, A., & Geetha,V. (2014, July).	Count of each token	Na¨ıve-Bayes 93.3%
4.	Detection of SQL Injection Attacks: A Machine Learning Approach Li, Q., Li , W., Wang, J., & Cheng, M. (2019).	Checking for the presence of special symbols, comment characters,semicolons, always true conditions, and the number of commands per statement	Ensemble Boosted and Bagged Trees classifiers 93.8%
5.	A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks Falor, A., Hirani, M., Vedant, H., Mehta, P., & Krishnan, D. (2022).	Assigns weights and biases to differentiate various aspects/objects from one another	CNN 94.84%

Table 9. Continuation of Table 8

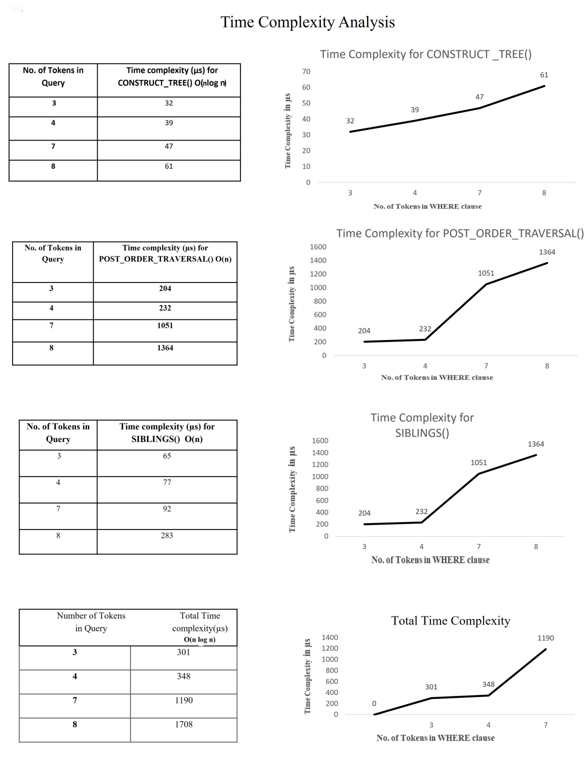
6.	Detection of SQL Injection based on artificial neural network Tang, P., Qiu, W., Huang, Z., Lian, H., & Liu, G. (2020).	Length of payload, number of Keywords, Sum of Keywords Weight,Number of spaces, Ratio of spaces, Number of special characters	LSTM 95%
7.	CODDLE: Code-Injection Detection With Deep Learning Abaimov, S., & Bianchi, G. (2019).	Each token is encoded as pair of numerical values <code, type> (C,T)	CNN 95.7%
8.	Deep Neural Network-Based SQLInjection Detection Method Zhang, W., Li, Y., Li, X., Shao, M., Mi, Y., Zhang, H., & Zhi, G. (2022).	TF-IDF of tokens	SQLNN 96.16%
9	SQL injection attack Detection using SVM Rawat, R., & Shrivastav, S. K. (2012).	Vector of strings formed from tokens of SQL query	SVM 96.47%
10	Detection of SQL InjectionAttack Using Machine Learning Basedon Natural Language Processing Triloka, J., Hartono, H., & Sutedi, S. (2022).	Frequency of the tokens using conditional probability	Naïve Bayes 97.5%
11	**Proposed Model Modified Parse -Tree Based pattern extraction Approach for detecting SQLIA using Neural Network Model**	Weighted Siblings pair extracted from the modified parse tree represents a pattern of WHERE clause	MLP 97.8%

Table 10. Scope of the proposed work

Sl. No	**Input Query**	**Siblings**	**Actual Label**	**Predicted Label**
1	Select title, phone, hire_date from employees where first name = 'Nancy' and last name = 'Edwards'.	(Column-String), (Column-String)	Legit	Legit
2	Select * from suppliers where supp id = 'SUP0145' and city='DELHI' or 1=1.	(Column-String), (Column- String), (Numeric-Numeric)	Inj	Inj
3	Select * from users where id = 1 or " ( )" or 1 = 1 – 1	(Column-Numeric), (String), (Numeric- Numeric)	Inj	Inj
4	Select count(*) from courses as c join attendance as a on c.course id = a.course id where c.course name = "English"	(Column-String), (Column-String)	Legit	Legit
5	Select length from river where length = (select max (length) from river);	(Column-Subquery)	Legit	Legit
6	Select name from student where marks>= (select max(marks) from results where dept = (select dept from institute where dept = (select dept from student where marks >= (Select max (marks) from results where dept='CA'))))	(Column-Subquery), (Column- Subquery), (Column- Subquery), (Column-Subquery), (Column String)	Legit	Legit

Table 11. Time complexity of the proposed work

Sl.no	Algorithm	Time complexity
1	CONSTRUCT TREE()	O(n log n)
2	POST ORDER TRAVERSAL()	O(n)
3	SIBLINGS()	O(n)
Total time complexity O(n log n)+O(n)+ O(n)= O(n log n)		



No. of Tokens in Query	Time complexity (µs) for CONSTRUCT_TREE() O(nlog n)
3	32
4	39
7	47
8	61

No. of Tokens in Query	Time complexity (µs) for POST_ORDER_TRAVERSAL() O(n)
3	204
4	232
7	1051
8	1364

No. of Tokens in Query	Time complexity (µs) for SIBLINGS() O(n)
3	65
4	77
7	92
8	283

Number of Tokens in Query	Total Time complexity(µs) O(n log n)
3	301
4	348
7	1190
8	1708

Figure 10. Time complexity

The proposed scheme's time complexity is estimated theoretically as O(n log n), where n is the number of tokens in the query's WHERE clause. Experiments have shown that increasing the number of tokens (n) increases the time complexity linearly. As a result, O(n log n) is proven and shown in Figure 10.

4.7 Comparative Analysis

Table 8 & Table 9 show the Comparative analysis between the existing models and the proposed MLP model. The existing models using techniques such as SVM, Logistic regression, Naive Bayes, Ensemble model, LSTM, and CNN mostly considered the SQL keywords and symbols as their features. But the proposed modified parse tree-based approach which considers the weighted siblings pair outperforms well than the existing models.

5 Conclusion

This paper has proposed a modified parse tree-based pattern extraction for SQLI detection implemented using the neural network. This model is plugged into an application server before the client request is submitted to the database server. Once it classifies the query as injected, the request should be aborted otherwise sent to the database server for response. Rather than using the pre-defined list of keywords and characters for attack detection, here, the query is transformed into a normalized modified parse tree, and patterns of the tree siblings are used for training the model with probability-based weights. Legitimate and injected queries differ significantly in patterns, and that helps classify them effectively. Based on this dataset, this proposed model identifies 22 pairs of unique siblings. They are trained with various ML models and a NN model MLP. The result of the evaluation phase shows that the proposed approach performs well with the NN model in all the metrics. Furthermore, this model can detect injected queries from its generalized form of patterns rather than by using a predefined list of keywords. Therefore, this model can be implemented with any SQL dataset to classify the query as legitimate or injected. In the future, it can be improved to detect attacks in different types of queries such as nested queries, queries with the union, and various types of SQLIAs.

References

[1] Injection attacks: The least glamorous attack is one of the most threateningsql injection attack: A major application security threat. https://securityintelligence.com/injection-attacks-the-least-glamorous-attack-is-one-of-the-most- threatening. Accessed: 2021-05-5.

[2] Sql injection attack: A major application security threat. urlhttps://www.kratikal.com/blog/sql-injection-attack-a-major- application-security-threat/. Accessed: 2021-05-5.

[3] Massive freepik data breach tied to sql injection attack. https://www.databreachtoday.com/massive-freepik-data- breach-tied-to-sql-injection-attack-a-14880. Accessed: 2022-05-3.

[4] Debabrata Kar, Suvasini Panigrahi, and Srikanth Sundararajan. Sqligot: Detecting sql injection attacks using graph of tokens and svm. *Computers & Security*, 60:206–225, 2016.

[5] Sayyed Mohammad Sadegh Sajjadi and Bahare Tajalli Pour. Study of sql injection attacks and countermeasures. *International Journal of Computer and Communication Engineering*, 2(5):539, 2013.

[6] Subhranil Som, Sapna Sinha, and Ritu Kataria. Study on sql injection attacks: Mode detection and prevention. *International Journal of Engineering Applied Sciences and Technology*, 1(8):23–29, 2016.

[7] MA Lawal, Abu Bakar Md Sultan, and Ayanloye O Shakiru. Systematic literature review on sql injection attack. *International Journal of Soft Computing*, 11(1):26–35, 2016.

[8] Bharti Nagpal, Naresh Chauhan, and Nanhay Singh. A survey on the detection of sql injection attacks and their countermeasures. *Journal of Information Processing Systems*, 13(4):689–702, 2017.

[9] Zainab S Alwan and Manal F Younis. Detection and prevention of sql injection attack: a survey. *International Journal of Computer Science and Mobile Computing*, 6(8):5–17, 2017.

[10] EE Ogheneovo and PO Asagba. A parse tree model for analyzing and detecting sql injection vulnerabilities. *West African Journal of Industrial and Academic Research*, 6(1):33–49, 2013.

[11] K Jhala and UD Shukla. Tautology based advanced sql injection technique a peril to web application. In *National conference on latest trends in networking and cyber security*, 2017.

[12] Rishiraj Saha Roy, Yogarshi Vyas, Niloy Ganguly, and Monojit Choudhury. Improving unsupervised query segmentation using parts-of-speech sequence information. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 935–938, 2014.

[13] Yong Fang, Jiayi Peng, Liang Liu, and Cheng Huang. Wovsqli: Detection of sql injection behaviors using word vector and lstm. In *Proceedings of the 2nd international conference on cryptography, security and privacy*, pages 170–174, 2018.

[14] Stanislav Abaimov and Giuseppe Bianchi. Coddle: Code-injection detection with deep learning. *IEEE Access*, 7:128617–128627, 2019.

[15] Ines Jemal, Omar Cheikhrouhou, Habib Hamam, and Adel Mahfoudhi. Sql injection attack detection and prevention techniques using machine learning. *International Journal of Applied Engineering Research*, 15(6):569–580, 2020.

[16] Romil Rawat and Shailendra Kumar Shrivastav. Sql injection attack detection using svm. *International Journal of Computer Applications*, 42(13):1–4, 2012.

[17] TP Latchoumi, Manoj Sahit Reddy, and K Balamurugan. Applied machine learning predictive analytics to sql injection attack detection and prevention. *European Journal of Molecular & Clinical Medicine*, 7(02):2020, 2020.

[18] Solomon Ogbomon Uwagbole, William J Buchanan, and Lu Fan. An applied pattern-driven corpus to predictive analytics in mitigating sql injection attack. In *2017 Seventh International Conference on Emerging Security Technologies (EST)*, pages 12–17. IEEE, 2017.

[19] B Kranthikumar and R Leela Velusamy. Sql injection detection using regex classifier. *Journal of Xi'an University of Architecture & Technology*, 12(VI):800–809, 2020.

[20] Abdalla Hadabi, Eltyeb Elsamani, Ali Abdallah, and Rashad Elhabob. An efficient model to detect and prevent sql injection attack. *Journal of Karary University for Engineering and Science*, 2022.

[21] Anamika Joshi and V Geetha. Sql injection detection using machine learning. In *2014 international conference on control, instrumentation, communication and computational technologies (ICCICCT)*, pages 1111–1115. IEEE, 2014.

[22] Mohammad Saiful Islam Mamun, Mohammad Ahmad Rathore, Arash Habibi Lashkari, Natalia Stakhanova, and Ali A Ghorbani. Detecting malicious urls using lexical analysis. In *Network and System Security: 10th International Conference, NSS 2016, Taipei, Taiwan, September 28-30, 2016, Proceedings 10*, pages 467–482. Springer, 2016.

[23] Lu Yu, Senlin Luo, and Limin Pan. Detecting sql injection attacks based on text analysis. In *3rd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2019)*, pages 95–101. Atlantis Press, 2019.

[24] Chamundeswari Arumugam, Varsha Bhargavi Dwarakanathan, S Gnanamary, Vishalraj Natarajan Neyveli, Rohit Kanakuppaliyalil Ramesh, Yeshwanthraa Kandhavel, and Sadhanandhan Balakrishnan. Prediction of sql injection attacks in web applications. In *Computational Science and Its Applications–ICCSA 2019: 19th International Conference, Saint Petersburg, Russia, July 1–4, 2019, Proceedings, Part IV 19*, pages 496–505. Springer, 2019.

[25] Qi Li, Weishi Li, Junfeng Wang, and Mingyu Cheng. A sql injection detection method based on adaptive deep forest. *IEEE Access*, 7:145385–145394, 2019.

[26] Joko Triloka, Hartono Hartono, and Sutedi Sutedi. Detection of sql injection attack using machine learning based on natural language processing. *International Journal of Artificial Intelligence Research*, 6(2), 2022.

[27] Naghmeh Moradpoor Sheykhkanloo. A learning-based neural network model for the detection and classification of sql injection attacks. *International Journal of Cyber Warfare and Terror-*

ism (IJCWT), 7(2):16–41, 2017.

[28] Peng Tang, Weidong Qiu, Zheng Huang, Huijuan Lian, and Guozhen Liu. Detection of sql injection based on artificial neural network. *Knowledge-Based Systems*, 190:105528, 2020.

[29] Wei Zhang, Yueqin Li, Xiaofeng Li, Minggang Shao, Yajie Mi, Hongli Zhang, and Guoqing Zhi. Deep neural network-based sql injection detection method. *Security and Communication Networks*, 2022, 2022.

[30] Ayush Falor, Manav Hirani, Henil Vedant, Priyank Mehta, and Deepa Krishnan. A deep learning approach for detection of sql injection attacks using convolutional neural networks. In *Proceedings of Data Analytics and Management: ICDAM 2021, Volume 2*, pages 293–304. Springer, 2022.

[31] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

A Meharaj Begum has completed Bachelor of Computer Science in 1999, Master of Computer Applications in 2002 and Master of Philosophy in 2005 from Manonmaniam Sundaranar University, Tamil Nadu, India. She has qualified SET in 2016 from Mother Teresa University for Women, Tamil Nadu and NET in2018 from University Grant Commission, New Delhi, India. She has seven years of teaching experience. Her areas of Interest include Data Base Management Systems, Data Structures and Alogorithms, Machine Learning, and Deep Learning Algorithms. She is currently doing her research work in NIT Tiruchirappalli,Tamilnadu,India.

Michael Arock is a Professor presently working in the Department of Computer Applications, National Institute of Technology, Tiruchirappalli. He graduated as a B.Sc.(Mathematics) from GTN Arts College, Dindigul, Madurai Kamaraj University and as an MCA from St.Joseph's College, Bharathidasan University and earned his Ph.D. from NITT under Bharathidasan University. His Doctoral thesis is on Design and Analysis of Parallel Algorithms on CREW PRAM and LARPBS models. His specialization is Parallel Algorithmics. His areas of Interest include Data Structures and Algorithms, High Performance Computing and Bioinformatics. So far, he has produced five doctorates in the field of DNA computing, Natural Language Processing and Bioinformatics. He has published 23 articles in reputed international journals, 19 in the proceedings of International Conferences and 6 in the proceedings of national conferences /seminars He concerns student counselling, motivating for better placements and designing value-based life-style. His hobbies are writing both classical and modern poems in Tamil, writing articles in Tamil and English and reading books, especially on Spiritualism.