Random Decision Forest approach for Mitigating SQL Injection Attacks

Pranjal Aggarwal*, Akash Kumar†, Kshitiz Michael‡, Jagrut Nemade§, Shubham Sharma ¶ and Pavan Kumar C ‡‡
Department of Computer Science and Engineering, Indian Institute of Information Technology Dharwad
Dharwad, Karnataka, India – 580009
*,†, ‡, §, ¶ – equal contribution
Email: *pranjalaggarwal1999@gmail.com, †akash5544kumar@gmail.com, ‡kshitizmichael007@gmail.com,
§jagrutjnemade@gmail.com, ¶shubhamm097@gmail.com, ‡‡pavankumarc@ieee.org

***Abstract*—Structured Query Language (SQL) is extensively used for storing, manipulating and retrieving information in the relational database management system. Using SQL statements, attackers will try to gain unauthorized access to databases and launch attacks to modify/retrieve the stored data, such attacks are called as SQL injection attacks. Such SQL Injection (SQLi) attacks tops the list of web application security risks of all the times. Identifying and mitigating the potential SQL attack statements before their execution can prevent SQLi attacks. Various techniques are proposed in the literature to mitigate SQLi attacks. In this paper, a random decision forest approach is introduced to mitigate SQLi attacks. From the experimental results, we can infer that the proposed approach achieves a precision of 97% and an accuracy of 95%.**

***Index Terms*—Ensemble classifier, Random Decision Forest and SQL injection attacks.**

I. Introduction

A Structured Query Language injection (SQLi) attack is also commonly referred to as a code injection attack [1], [2]. It is a security vulnerability which allows access to data which is intended to be protected and inaccessible to unauthorized personnel. This technique is used by hackers to inject (as in force or drive data) into an existing database by exploiting (security) vulnerabilities in the structure/implementation of the database. This is done by placing malicious code in the field which accepts input linked to SQL queries through a web page. The exploit can be used by the attacker in many ways, such as *spoofing someone's identity, tinker sensitive data, expose the entire dataset* to name a few.

Structured Query Language injection (SQLi) was first observed in 1998 and yet after 22 long years of its discovery, still remains a top database security concern [1], [2]. The Open Web Application Security Project (OWASP), a non-profit organisation that works to improve the security of the software, has frequently placed injection attacks at the first place among the top 10 Web Application Security Risks [3]. This makes it an intriguing and challenging issue to address. There have been instances where even the technology giants could not defend themselves from the attack. Some recent famous examples of this kind of attack are Google Cloud Database(Aug 2020) [4], Cisco(Aug 2020) [5], IBM (Mar 2020) [6] and US Election Assistance Commission (2016) [7].

978-1-6654-2849-1/21/$31.00 © 2021 IEEE

Various researchers have attempted to address the issue of mitigating SQL injection attacks either at the software level by developing web vulnerability identification tools or by analyzing the SQL queries fed to the system [8]–[15]. Antunes and Vieira [16] have developed tests to identify the SQL injection attacks present in the web services comparing injection attacks with the normal behaviour (or execution) of SQL queries gathering information on operations, call parameters and input domains. Uwagbole et al. [17] have identified some of the patterns in the query as attacks and if such patterns are found then those queries are labeled either as SQLIA positive (with a value 1) or SQLIA negative (with a value 0). They have classified such labeled data using a two class support vector machine to check whether the queries are injection attacks or normal queries at the query injection point itself.

In this paper, the problem of identification of possibly unsafe SQL query statements is addressed using random forest classifier approach, an ensemble learning technique [18]. This paper is organized as follows. In section II preliminaries required to understand and design of the proposed model is discussed. Proposed model is introduced in section III. Result analysis is given in section IV and conclusions drawn are given in sections V.

II. Preliminaries

A SQL injection (SQLi) is a common attack technique that uses malicious SQL code in the form of queries for backend database access and manipulation to derive information that was not intended to be displayed or published.

Common types of SQLi include:

- **In-band:** It is a simple type of SQL injection which occurs when the attacker uses the same communication channel to launch the query and get the result. *Example: end of line comment (@var select @var as var into temp end −−), tautology (1 or 1 = 1), etc.*
- **Inferential:** It is also known as blind SQLi. No actual transfer of data takes place in this type via the web application. Although the result of the attack is

not visible but instead the structure of the database could be reconstructed by sending payloads, observing responses, the behaviour of database, etc. *Example:* $'utl\_http.request('http://192.168.1.1/')'$

- **Out of band:** This attack comes into play when the server responses are not very stable (making an inferential attack unreliable). The channel used for attacking and gathering results is usually different.

A. Decision Tree

A decision tree is a simple tree-like structure where at each stage the data is sub-divided into small data units based on the features of the datasets [19], [20]. The feature to be used at each stage is decided on several factors such as entropy, information gain, gini impurity, etc. Based on this, the data is separated into different classes which are further used for the prediction. These decision trees are the building blocks of the Random Forest model.

B. Random Forest

Random Forest is an ensemble learning method for classification and regression [21], [22]. It consists of a large number of individual decision trees where each individual tree predicts a certain class and the predicted class with the most votes becomes the prediction outcome. In simple, random forest technique is majority voting approach over large number of trees in a forest.

Breiman [21] define random forest approach of classifying as follows: A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(x, \Theta_k), k = 1, \ldots\}$ where the Θ_k are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x.

III. Proposed Model

A SQL statement is a collection of keywords and expressions. If a SQL statement can be identified as a potential risk before its execution then such attacks can be prevented. In this proposed approach, random forest ensemble learning is used to identify the possible SQLi attack statements.

In the first stage of the identification of a statement as safe or not, a score will be assigned to each query according to the characters, numbers and types of keywords and expressions present in it. Each keyword is scored on the basis of their potential risk factor on a scale of 3, with 1 being the least unsafe and 3 the most unsafe from the knowledge of previous SQL injection attack types.

A. Analyzing the Dataset

To proceed with the above-mentioned approach, a dataset [23] was obtained from Kaggle in which several SQL queries were labelled and listed. All those queries which resulted in a successful penetration were labelled as 1 and others were labelled as 0.

B. Deciding the Features for the Model

The features decided for the model include:

- **Comment Character:** Includes characters like '/*', '- -', etc. with a score of 1. They can be used to comment out a portion of the query or override it.
- **No. of semicolon:** Each was given a score of 1. Each statement is terminated by a semicolon. Thus, it gives the number of commands passed by the client.
- **Logical Operators:** Includes logical keywords like 'or', 'null', etc.
- **True Conditions:** Includes conditions like "1 = 1", "1 || 2" which are always true.
- **Keywords:** Includes keywords related to SQL. Scores are obtained from the spreadsheet we created by scoring each keyword [24].
- **Wildcard characters:** Number of wildcard characters present in the statement.

A sample list of scores assigned to the keywords and logical operators present in SQL is given in Table I. All those terms which were not included in the spreadsheet prepared for the keywords [24] and wildchar [25] were ignored.

C. Pre-processing the Dataset

- A list of keywords which are used in SQL queries was prepared by collecting those from several resources [26], [27] (and by adding a few of them while scoring the dataset).
- This was followed by scoring all of these keywords according to their risk factors and producing a spreadsheet [24] with keywords and their respective scores.
- Further, wildcard characters, logical operators and comment characters were separately scored which resulted in another spreadsheet [25].
- Some of the queries from the dataset were scored according to the features and the scores of the respective keywords and wildcard characters.

D. Training of Model

The whole dataset was statistically inferred using graphs which facilitated in checking of how one feature varies with respect to other features. Graph in Fig. 1 gives the variation of one feature with the other.

A correlation matrix was built to determine the correlation between the features with each other. The heatmap in Fig. 2 gives the level of correlation between the features. Upon these examinations, a Random Forest Classifier is implemented after performing a 10-fold cross validation on the data set created. A confusion matrix was also used to verify the False Positives and False Negatives.

E. Reproducible result

The source code for this work is freely available online for the readers/researchers to use it for research[1].

[1]https://github.com/T-I-P/SQLi

TABLE I
SAMPLE SCORES ASSIGNED FOR THE SQL KEYWORDS AND LOGICAL OPERATORS

SQL Keyword	Score Assigned	SQL Keyword	Score Assigned
CREATE	3	DATABASE	3
CROSS	2	DBCC	1
CURRENT	2	DEALLOCATE	1
CURRENT_DATE	3	DECLARE	1
CURRENT_TIME	1	DEFAULT	2
FROM	1	DELETE	3
FULL	2	DENY	3
CURSOR	2	DESC	1
DUMP	2	DISK	2
END	1	DISTINCT	2
DROP	3	DOUBLE	1
1 = 1	2	*	1

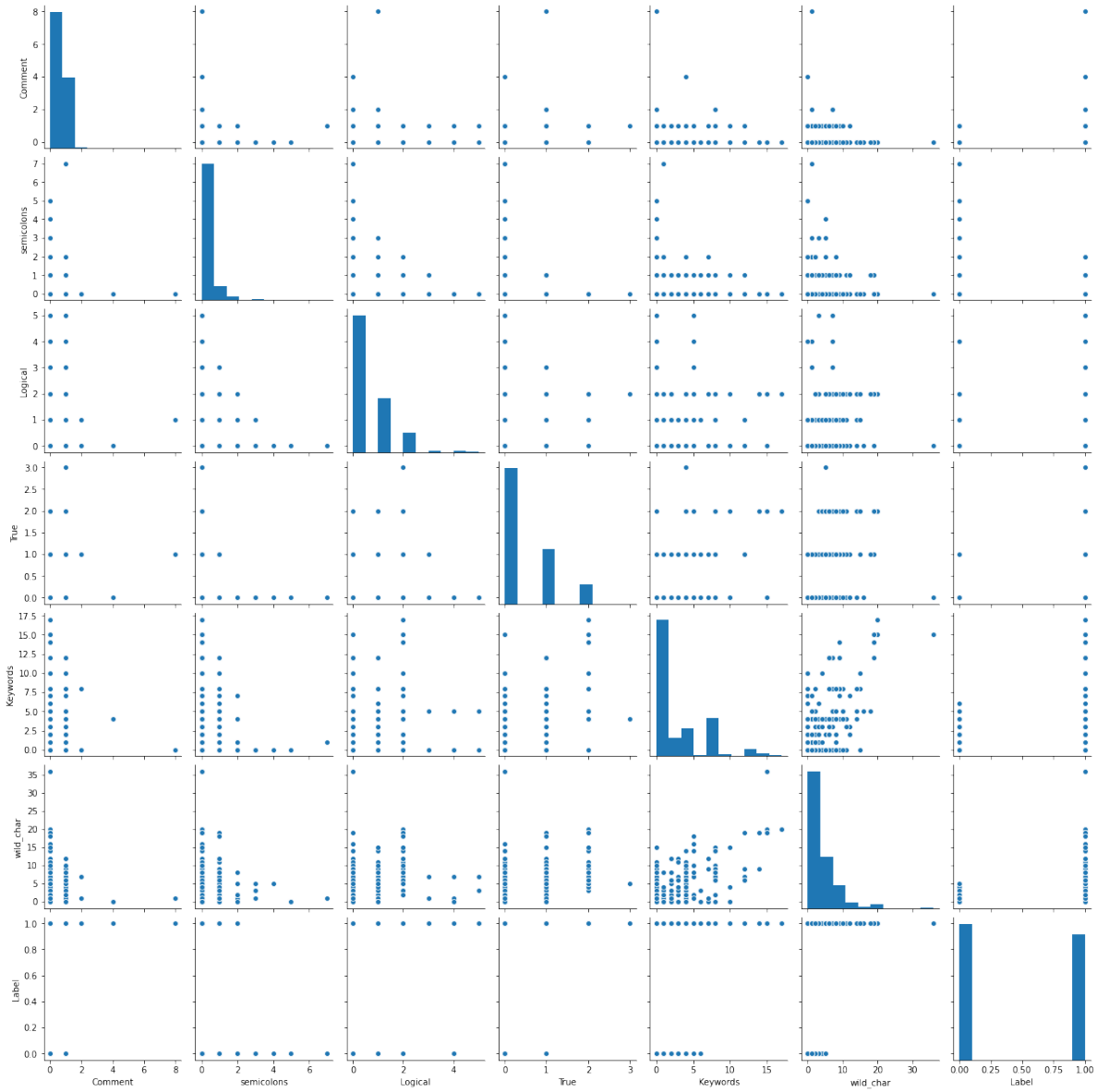

Fig. 1. Variation of one feature with other

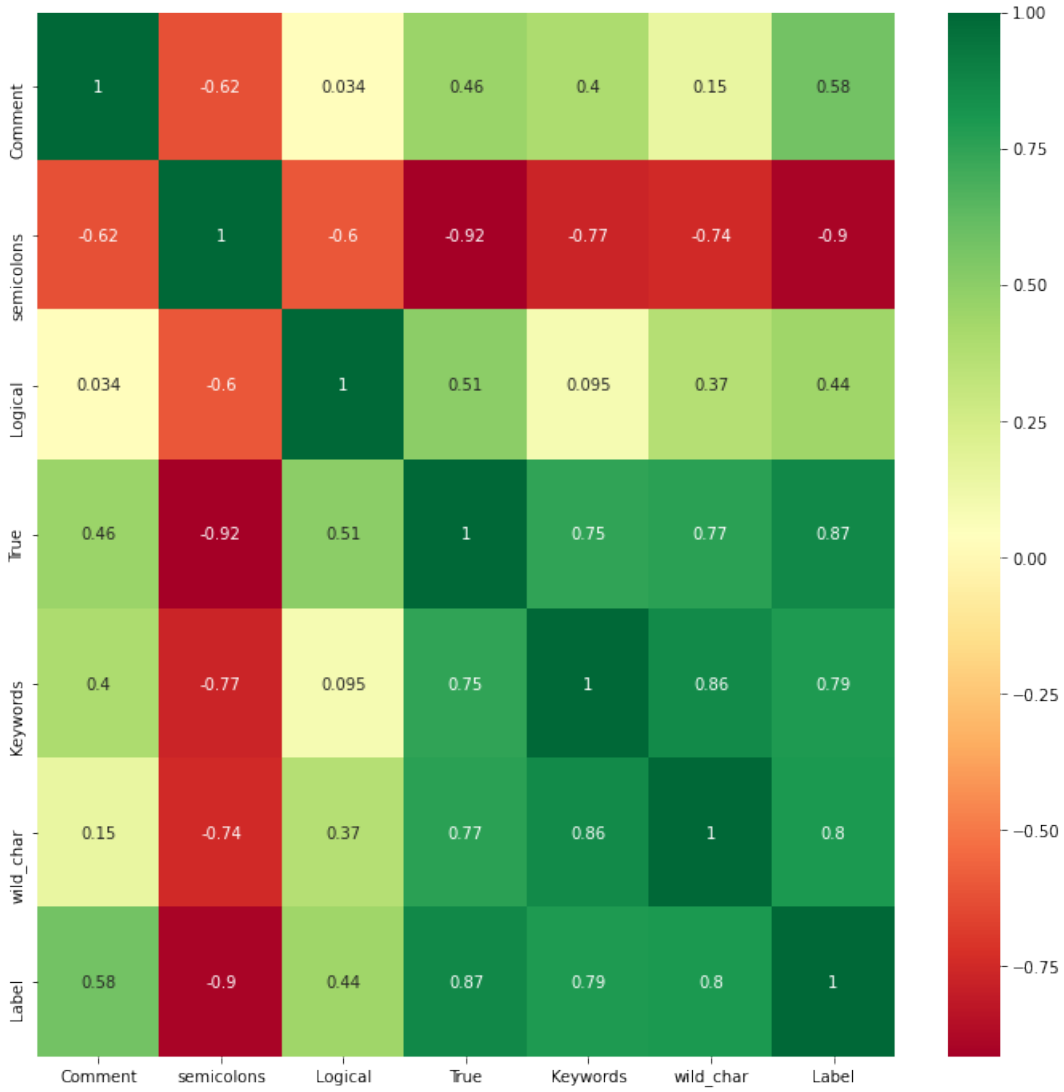

Fig. 2. Correlation between the features

IV. Result Analysis

The proposed method was tested on the data set comprising of 578 SQL statements. The following confusion matrix is obtained from the implementation of the proposed approach.

$$\begin{bmatrix} \text{True Positive (TP)} & \text{False Positive (FP)} \\ \text{False Negative (TN)} & \text{True Negative (TN)} \end{bmatrix} = \begin{bmatrix} 260 & 8 \\ 17 & 293 \end{bmatrix}$$

Precision, Recall and Accuracy values of the proposed method is as follows:

$$Precision = \frac{\text{TP}}{\text{TP + FP}} = 0.9701$$

$$Recall = \frac{\text{TP}}{\text{TP + FN}} = 0.9386$$

$$Accuracy = \frac{\text{TP + TN}}{\text{TP + TN + FP + FN}} = 0.9567$$

V. Conclusion

Identifying and mitigating potential SQLi attacks are of high importance as it can prevent possible theft or modification of sensitive data stored in relational databases. In this paper, random forest ensembler based classification is implemented to address the issue of classifying SQL injection attacks. From the results we can infer that the model was able to identify potentially unsafe SQL queries with an accuracy of 95%. This accuracy is quite good pertaining to the size of the dataset currently used. It is planned to enhance the dataset in terms of its size and diversity, and observe its effects on the proposed model.

References

[1] R. Jahanshahi, A. Doupé, and M. Egele, "You shall not pass: Mitigating sql injection attacks on legacy web applications," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 445–457.

[2] J. Hu, W. Zhao, and Y. Cui, "A survey on sql injection attacks, detection and prevention," in *Proceedings of the 2020 12th International Conference on Machine Learning and Computing*, 2020, pp. 483–488.

[3] OWASP. (2017) OWASP Top Ten Web Application Security Risks | OWASP. [Online]. Available: https://owasp.org/www-project-top-ten/2017/Top_10.html

[4] offensi.com, "How to Contact Google SRE: Dropping a Shell in Cloud SQL - Offensi," august 2020. [Online]. Available: https://offensi.com/2020/08/18/how-to-contact-google-sre-dropping-a-shell-in-cloud-sql/

[5] A. Hashim. (2020, august) Cisco Addressed Multiple Bugs In Data Center Network Manager. [Online]. Available: https://latesthackingnews.com/2020/08/02/cisco-addressed-multiple-bugs-in-data-center-network-manager/

[6] IBM, "Security Bulletin: SQL Injection Vulnerability Affects IBM Sterling B2B Integrator EBICS (CVE-2019-4597)," 2020. [Online]. Available: https://www.ibm.com/support/pages/security-bulletin-sql-injection-vulnerability-affects-ibm-sterling-b2b-integrator-ebics-cve-2019-4597

[7] Z. Zorz. (2017, feb) Hacker breached 60+ unis, govt agencies via SQL injection - Help Net Security. [Online]. Available: https://www.helpnetsecurity.com/2017/02/16/hacker-govt-agencies-via-sql-injection/

[8] J. Fonseca, M. Vieira, and H. Madeira, "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks," in *13th Pacific Rim international symposium on dependable computing (PRDC 2007)*. IEEE, 2007, pp. 365–372.

[9] R. Amankwah, J. Chen, P. K. Kudjo, and D. Towey, "An empirical comparison of commercial and open-source web vulnerability scanners," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1842–1857, 2020.

[10] Z. Djuric, "A black-box testing tool for detecting SQL injection vulnerabilities," in *2013 Second International Conference on Informatics & Applications (ICIA)*. IEEE, 2013, pp. 216–221.

[11] L. Saoudi, K. Adi, and Y. Boudraa, "A rejection-based approach for detecting SQL injection vulnerabilities in web applications," in *International Symposium on Foundations and Practice of Security*. Springer, 2019, pp. 379–386.

[12] M. S. Aliero, K. N. Qureshi, M. F. Pasha, A. Ahmad, and G. Jeon, "Detection of structure query language injection vulnerability in web driven database application," *Concurrency and Computation: Practice and Experience*, p. e5936, 2020.

[13] L. Ntagwabira and S. L. Kang, "Use of query tokenization to detect and prevent SQL injection attacks," in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 2. IEEE, 2010, pp. 438–440.

[14] G. Deepa and P. S. Thilagam, "Securing web applications from injection and logic vulnerabilities: Approaches and challenges," *Information and Software Technology*, vol. 74, pp. 160–180, 2016.

[15] R. Chandrashekhar, M. Mardithaya, S. Thilagam, and D. Saha, "SQL injection attack mechanisms and prevention techniques," in *International Conference on Advanced Computing, Networking and Security*. Springer, 2011, pp. 524–533.

[16] N. Antunes and M. Vieira, "Detecting SQL injection vulnerabilities in web services," in *2009 Fourth Latin-American Symposium on Dependable Computing*. IEEE, 2009, pp. 17–24.

[17] S. O. Uwagbole, W. J. Buchanan, and L. Fan, "Applied machine learning predictive analytics to SQL injection attack detection and prevention," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 1087–1090.

[18] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *The journal of machine learning research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[19] J. R. Quinlan, "Simplifying decision trees," *International journal of man-machine studies*, vol. 27, no. 3, pp. 221–234, 1987.

[20] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[21] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[22] T. Yiu. (2019, jun) Understanding random forest: How the algorithm works and why it is so effective. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2/

[23] S. S. H. Shah, "SQL injection dataset | Kaggle," may 2020. [Online]. Available: https://www.kaggle.com/syedsaqlainhussain/sql-injection-dataset

[24] P. Aggarwal, A. Kumar, K. Michael, J. Nemade, and S. Sharma, "SQLi Keywords scores," 2020. [Online]. Available: https://github.com/T-I-P/SQLi/blob/master/Keywords_Scores.xlsx

[25] ——, "SQLi Logical-Operators," 2020. [Online]. Available: https://github.com/T-I-P/SQLi/blob/master/Logical-Operators_True-Conditions__Scores.xlsx

[26] W3Schools, "SQL Keywords Reference." [Online]. Available: https://www.w3schools.com/sql/sql_ref_keywords.asp

[27] Dofactory, "SQL Server Keyword - Dofactory." [Online]. Available: https://www.dofactory.com/sql/keywords