A Clustering-Based Approach for Effective Prevention of SQL Injections

Smit P Shah, Achyuta Krishna V, V.L. Kartheek, and R. Gururaj

BITS Pilani, Hyderabad
`f20170080h@alumni.bits-pilani.ac.in`,
`f20180165@hyderabad.bits-pilani.ac.in`,
`p20210105@hyderabad.bits-pilani.ac.in`,
`gururaj@hyderabad.bits-pilani.ac.in`

Abstract. SQL injection attack happens due to insertion of a malicious SQL query into the application server via the input data from the attacker. A successful SQL injection attack can read and/or modify sensitive data stored in the database, execute administrative commands on the database such as DB shutdown, and in some cases issue harmful commands to the operating system. Owing to this, injection based security vulnerabilities rank in the top ten security pitfalls identified by Open Web Application Security Project. Due to the significance of the security of web-based database applications, prevention of the SQL injection at the right time is of paramount importance. A lot of research has been carried out on SQL injection attack detection and prevention. However, there is not much work done on clustering of SQL injections into different types. Since different types of SQL injections can be countered effectively using different techniques, grouping of SQL injections can help organizations to allocate resources judiciously for effective prevention of SQL injection attack. In this paper, we present an empirical study to identify the best suited unsupervised machine learning algorithm for clustering SQL injections, with an intention to find the frequency of occurrence of different types of injections. This approach would aid organizations to strengthen their security measures against particular types of injections in a more focused way. We also give a detailed analysis of efficacy of the clustering algorithms used.

Keywords: SQL injection · unsupervised learning · detection · prevention · clustering.

1 Introduction

Database security is a vital part of information security. It is estimated that significant number of web application attacks are caused by SQL injections. Recent SQL injection attacks include- the 2020 data breach of millions of username and passwords [1], the 2020 attack to steal credit card data [2] and the 2017 attack on more than 60 universities and governments worldwide [2]. Gaining unauthorized access to the enterprise databases allows attackers to have a greater control over pivotal data. This allows attackers to do nearly anything with the data, including alteration and deletion of data. The SQL Injection Attack (SQLIA) will occur when the malicious SQL query is received at the server through input forms. As blazoned by the Open Web Application Security Project (OWASP) association, injection attack has been the leading security threat in 2013 and 2017, and SQL injection attack is one of the most prominent type [3]. Hence, mitigating these injections using robust solutions becomes of utmost importance otherwise there is a serious threat to the entire community. In this work, we propose to use an unsupervised learning approach for clustering SQL injections so that organizations can minimize the impact at the onset of the issue itself.

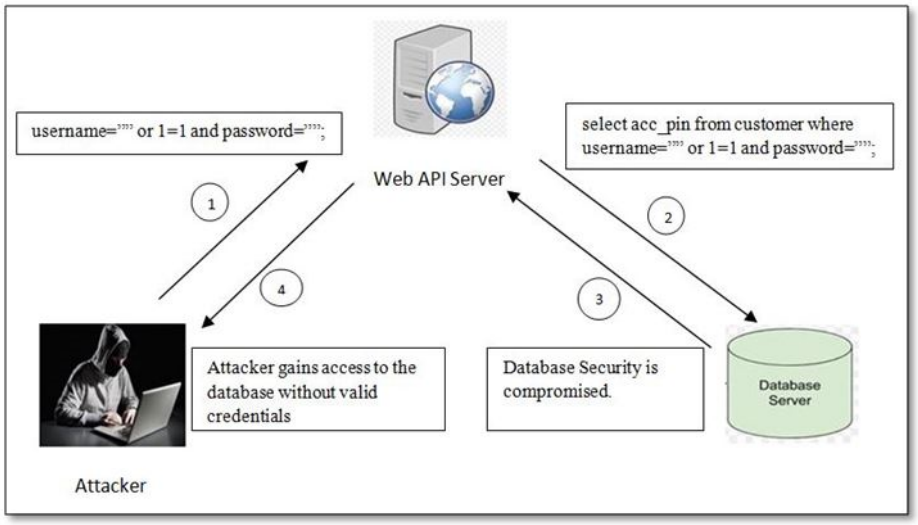

username="" or 1=1 and password="";
Web API Server
select acc_pin from customer where username="" or 1=1 and password="";
1
2
3
4
Attacker gains access to the database without valid credentials
Database Security is compromised.
Database Server
Attacker

Fig. 1. Illustration of an SQL injection attack

1.1 SQL Injection Attack

SQL stands for Structured Query Language. It is a language used for querying and modifying data in a relational database. SQL Injection is a malicious query statement formulated by an attacker with an intent to extract or update records in the database. For example, consider the following query.
select studentMarks, studentid, studentName, FROM students where studentName = 'Wiley' and studentId=12

If an attacker were to input value of studentName as a string- *Wiley' OR 1=1–*. The resultant query would be-
select studentMarks, studentid, studentName, FROM students where studentName = 'Wiley' OR 1=1—' and studentId=12

This query would return marks of every student in the database.

Though there exist multiple types of SQL injections, in this work we have considered the following important types.

- Union based injection: The attacker uses the UNION keyword to concatenate a malicious query to the original query.
- Tautology based injection: In tautology, authentication is bypassed by injecting malicious input so that one or more conditional statements is intended to be true.
- Logically Incorrect query: In this attack, intentionally the attacker tries to throw an error from the database to understand the database schema.
- Timing based injection: In this attack, the aim is to delay database responses by using keywords like- *sleep, wait, etc.*
- Alternately encoded injection: In this type, the injection query is hidden by masking it with ASCII / hexadecimal characters.

For instance, in Fig. 1, the attacker bypassed authentication on database server resulting in access to the database because the submitted SQL query always evaluated to be true by adding one or more conditional expressions such as 1=1. This is a simple example of tautology based SQL injection attack.

1.2 How does SQLIA affect the Database security

An SQLIA can compromise database security in multiple ways. The number of SQLIAs have increased drastically in the recent past. As per a report, these attacks had accounted for almost half of all web operation attacks in 2017 [4]. SQL injection attacks compromise the security, confidentiality, privacy and integrity of the database. This might lead to a financial impact for the organisations or a data breach, ultimately leading to a huge loss of trust among the customers.

In the recent past, researchers have attempted to detect SQL injections through machine learning models which enabled organisations to take countermeasures [5-7]. But, we observed that there is less research work carried out on clustering the injections. In the work presented in this paper, we conduct an empirical study on three unsupervised learning clustering algorithms to identify the one which is best suited for clustering SQL injection attacks.

The rest of paper is organized as follows. We discuss related work along with motivation for our proposal in Section 2. In Section 3, we give a detailed account of our proposal along with methodology. Section 4 gives analytical insights of the clustering algorithms used. Finally in Section 5, we conclude the paper.

2 Related work

There has been extensive research on using machine learning and deep learning based techniques to classify SQL statements into injections and non-injections. Kevin Zhang uses various machine learning algorithms such as random forest, SVM and deep learning based algorithms like convolutional neural networks and multilayer perceptron to detect the presence of an SQL injection [5]. D. Tripathy et al. classify SQL statements into injection and non-injection by the use of deep learning techniques like Adaboost and Deep ANN [6]. Umar Farooq uses an ensembling of gradient boosting machine (GBM), Adaboost, Extended gradient boosting machine (XGBM) and Light gradient boosting machine (LGBM) to detect SQL injections [7]. Wang et al. have tried to establish a detection method based on static and dynamic analysis approach [8]. Kindy et al. in their survey paper have tried to categorize SQL injections and present the prevention techniques [9]. They have done a comparative analysis of the prevention techniques and presented details of the different schemes and defense mechanism against each kind of SQL injection. Similarly, Halfond et al. have tried to compare which techniques are effective against which kind of SQL injections [10]. Omar Kasim et al. in their work have provided an ensemble approach for detecting the severity level of SQL injections into three broad categories: simple, unified or lateral attack [11]. Courant proposes a developer friendly approach using abstract syntax trees which helps developers create dynamic queries with ease to prevent developers from unknowingly writing a malicious SQL injection [12]. Gandhi et al. propose a convolutional neural network based solution based on CNN-BiLSTM for detecting SQL injections accurately. It provides a comparison between traditional CNN, LSTM and Bi-LSTM and concludes with Bi-LSTM as the winner [13]. Hirani et al. propose another deep learning based solution where they compare multiple algorithms for detection of SQL injection and conclude that CNN is the best amongst all [14]. Majority of the work is aligned towards detection of a SQL injection. There is no concrete work in classifying them into different types. From the research work carried out by Jemal et al. it is evident that there exist multiple techniques for prevention of SQL injections and it is also clear that a given technique is more suitable for preventing specific type of attacks [15].

We observe that majority of research work carried out so far in this area focus more on detection of presence of SQL injection attack rather than finding the type of attack. The main problem with this approach is that some organizations might have spent most of their resources on deploying a particular technique, which might not be an effective solution against handling other injection types. On the other hand some organizations might have deployed all counter-measures which would result in ineffective utilization of resources because some of the types of attacks they have considered may not be frequent ones. Hence, we propose that if an organization can get information about the frequency of occurrence of different types of attacks it would help in optimal and effective allocation of resources to prevent SQL injection attacks. Owing to above reasons, in this paper we propose a machine learning based model to cluster SQL injections that have occurred, which would enable the organizations to allocate resources and spend more developer cycles in an effective way to counter SQL injections.

3 Proposed work

With the motivation explained in the previous section, we propose to conduct a detailed study on three unsupervised machine learning algorithms for clustering SQL injections. Now we present the overall process involved, as depicted in Fig. 2.

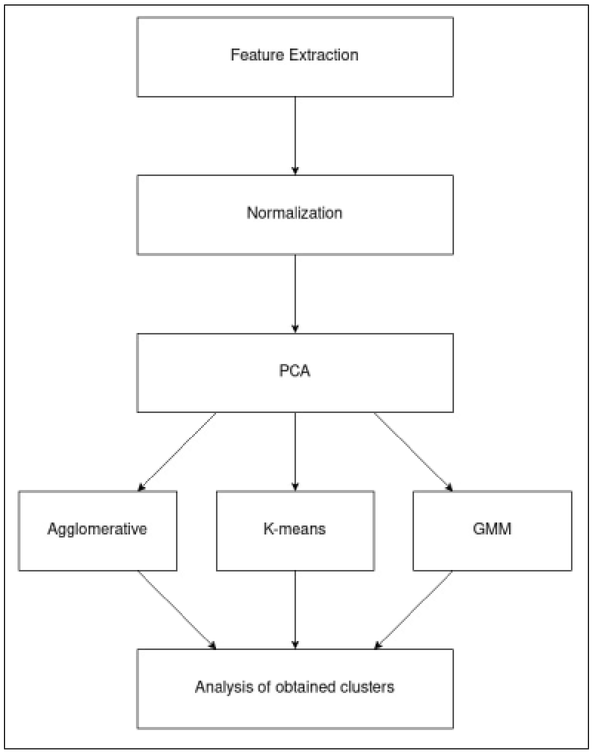


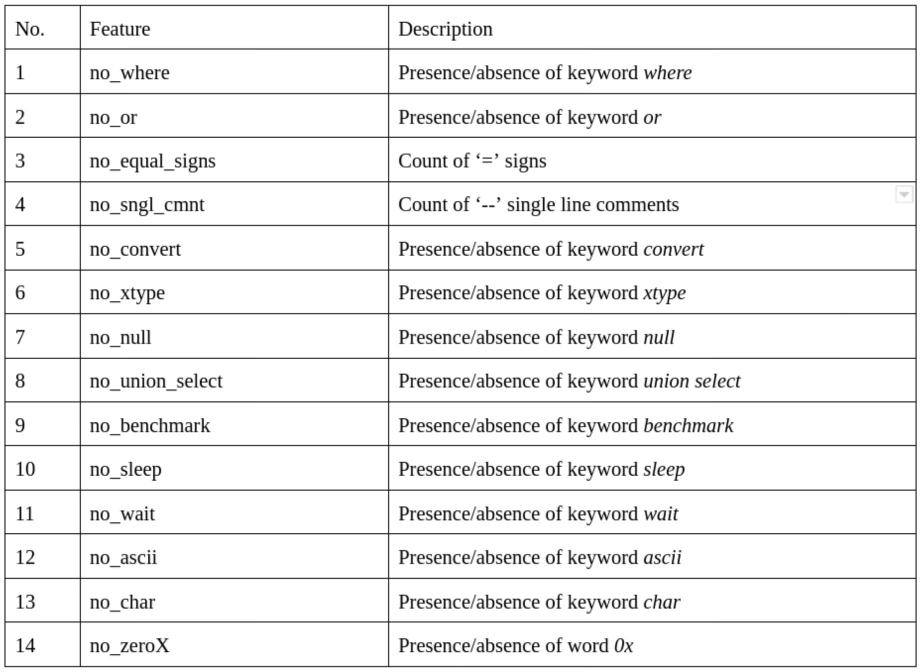
Fig. 2. Flowchart illustrating our proposed methodology

3.1 Data collection

We chose an existing Kaggle dataset of 30873 points and performed unsupervised learning on the same [16]. The dataset consists of around 38% SQL injections and the rest are not SQL injections. We have filtered out the SQL injections from the dataset and then extracted features from it.

3.2 Preprocessing the dataset

Feature extraction Based on our domain knowledge about the types of SQL injections, we have considered 14 features as given in Fig. 3.



No.	Feature	Description
1	no_where	Presence/absence of keyword *where*
2	no_or	Presence/absence of keyword *or*
3	no_equal_signs	Count of '=' signs
4	no_sngl_cmnt	Count of '--' single line comments
5	no_convert	Presence/absence of keyword *convert*
6	no_xtype	Presence/absence of keyword *xtype*
7	no_null	Presence/absence of keyword *null*
8	no_union_select	Presence/absence of keyword *union select*
9	no_benchmark	Presence/absence of keyword *benchmark*
10	no_sleep	Presence/absence of keyword *sleep*
11	no_wait	Presence/absence of keyword *wait*
12	no_ascii	Presence/absence of keyword *ascii*
13	no_char	Presence/absence of keyword *char*
14	no_zeroX	Presence/absence of word *0x*

Fig. 3. Feature set

These are the fourteen features which we have tried to extract for each data point. Except feature number three and four, rest all are Boolean values. The keywords *sleep*, *wait* and *benchmark* are typically used to delay the execution of queries, which is a characteristic of timing-based injection attacks.

The keyword *union select* represents the number of unions followed by a select keyword which characterizes a union based injection. The keyword *ASCII, char, 0x* denote the presence of conversion of a string to other type, and can be considered an indicator for alternately encoded injection. One way of generating a logically incorrect query is by converting a row object into another primitive data type. This can be achieved using keywords *convert* and *xtype* .

Normalizing the dataset Once we have the features extracted for each of the data points, we normalize the dataset. Normalization helps in transforming our attribute values to a similar scale. This will help in achieving uniformity and can improve the performance of the model. We have considered Z-score normalization which helps to ensure that our mean is 0 and standard deviation is 1. The Z-score is calculated as :

$$Z = \frac{(x - mean)}{std.\ deviation} \tag{1}$$

Principal Component Analysis Once we have normalized the dataset, we apply Principal Component Analysis (PCA) on the features. The advantage of using PCA is that it reduces the number of features, while preserving maximum information about the dataset. From the initial 14 features we reduce it to 8, preserving about 75% of the information. Smaller datasets are easier to explore and models are less expensive in terms of computation.

3.3 Learning the patterns in the dataset

After pre-processing the dataset, we apply clustering algorithms to it. We have used three unsupervised machine learning algorithms. Unsupervised learning involves learning from the information which is neither classified nor labeled. The model achieves the intended goal of clustering from the available information without any guidance. Clustering is one important aspect of unsupervised machine learning which is meant for grouping similar objects to form clusters.

We have used the following algorithms in our study:

- Agglomerative clustering- It is a type of hierarchical clustering algorithm wherein initially each point is an individual cluster and further each cluster is merged until all points fall in one big cluster. It is a bottom-up clustering algorithm.
- K-Means- It is a type of unsupervised clustering algorithm where the k value is predefined to be the number of clusters and each point is assigned to the nearest center. All such points club together to form a cluster.
- Gaussian Mixture Model (GMM)- It involves the mixture of multiple probability distributions and it can accommodate clusters that have different sizes and shapes.

3.4 Optimal number of clusters

The goodness of clustering depends on two main factors: the points within a cluster should be as similar as possible and the points in different clusters should be as distinct as possible. i.e., inter cluster distance should be high and intra-cluster distance between points should be low. One statistical measure to find the goodness of clustering algorithms is *Silhouette coefficient.*

Silhouette coefficient (S) in simple terms can be explained as:

$$S = \frac{(b-a)}{max\,(a,b)} \tag{2}$$

where, a is average intra-cluster distance and b is average inter-cluster distance

The value of the Silhouette coefficient ranges from -1 to +1 where +1 means that the clusters are separate and distinguishable, 0 means that the clustering is decent but there is scope for improvement and -1 means that the clustering is incorrect.

For each of the three models, we computed the Silhouette coefficient to determine the optimal number of clusters and obtained the following results. Agglomerative clustering returns the optimal number of clusters to be 5 with Silhouette coefficient of 0.41 as depicted in Fig. 4. K-Means returns a Silhouette coefficient score of 0.42 for 3 clusters as seen in Fig. 5. GMM returns the optimal number of clusters as 2, with the Silhouette coefficient for the same being 0.42 as shown in Fig. 6.

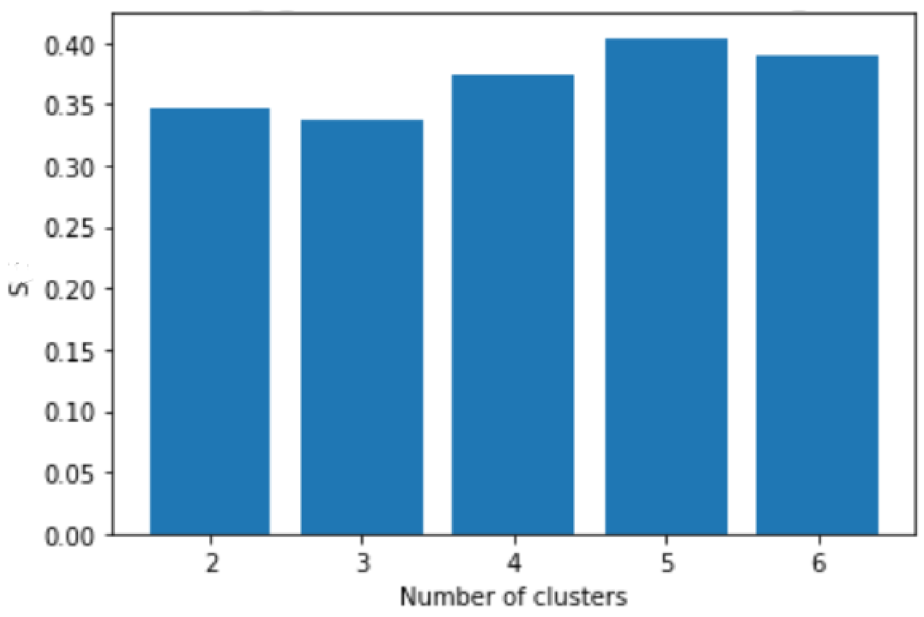


Fig. 4. Silhouette coefficients for Agglomerative clustering

4 Analysis

We analyzed the different clusters formed post which we attempted to map the clusters formed to a few specific type of SQL injections. Based on the values of specific features for each cluster, we determined the presence or absence of a particular injection type in the cluster. The results for GMM were not very promising with 2 being the optimal number of clusters. K-Means algorithm returned 3 as the optimal clusters but since this algorithm could not represent some of the types, we had to drop it. The 5 clusters obtained from agglomerative clustering gave more useful insights than the other two.

On analyzing the results for agglomerative clustering which can be seen in Fig. 7 we found the following. All 318 points in cluster 1 were timing based

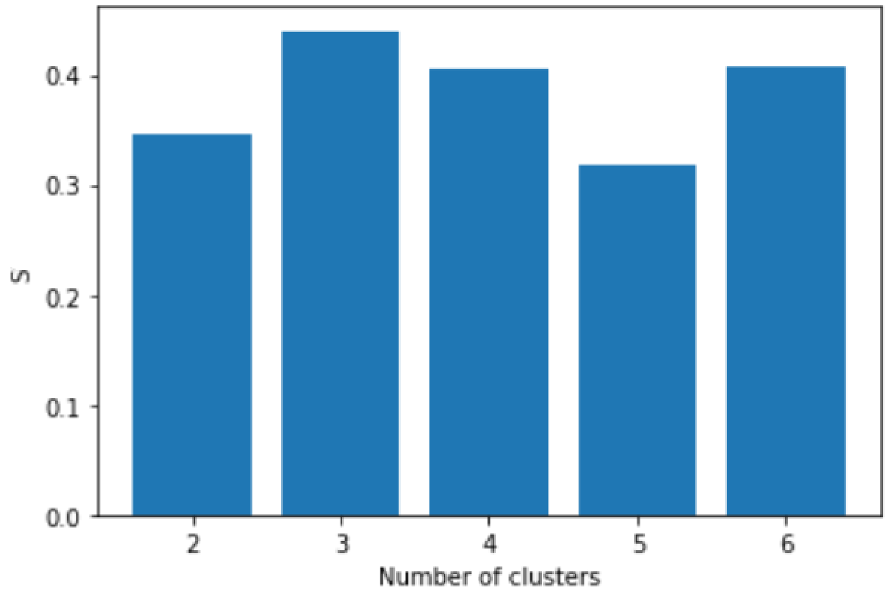


Fig. 5. Silhouette coefficients for K-means

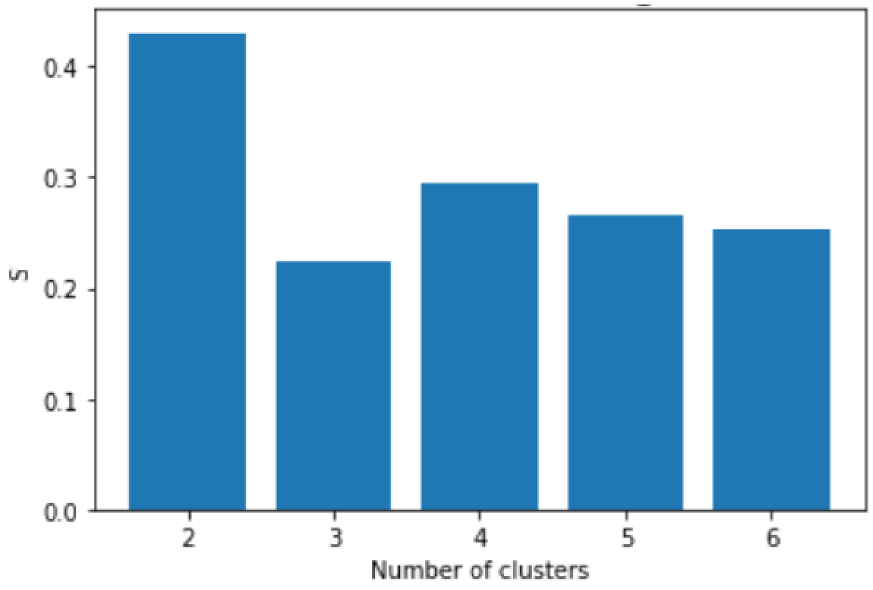


Fig. 6. Silhouette coefficients for Gaussian Mixture Model

injections with some being alternately encoded. All points in cluster 2 were union based injections out of which 112 points were alternately encoded. Cluster 3 had a majority of alternately encoded injections and 108 of them being logically incorrect. Cluster 4 contained the maximum number of points and it contained a mix of timing based injections and alternately encoded injections. Cluster 5 had only 6 points, all of which seem to be outliers in the dataset.

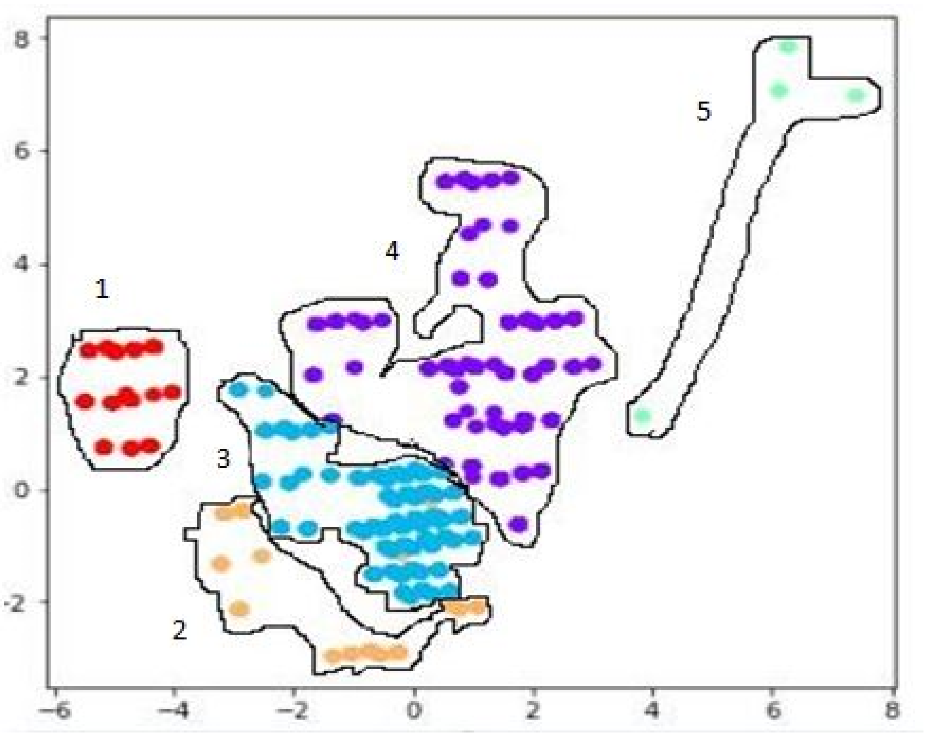


Fig. 7. Scatter plot with clusters

The attributes *no_ sleep*, *no_ wait* and *no_ benchmark* were used to identify timing-based injection attacks. On analyzing the data points that have a presence of these keywords, it was found that all the points in cluster 1 had the presence of either one of the three keywords corresponding to timing-based attacks. It is possible for an SQL injection to be of multiple types. For example, a SQL query can be both union based injection and alternately encoded. This can be seen from the results , where cluster 2 consists of SQL statements which represent both union based and alternately-encoded SQL injections. Similarly, the attributes *no_ convert* and *no_ xtype* were used for identification of logically incorrect queries. The attribute *no_ union_ select* and *no_ null* were used to group union based injections. For alternately encoded injections *no_ ascii*, *no_ char*, *no_ zeroX* were used to detect alternately encoded injections.

Apart from Silhouette coefficient, we further verified our algorithms with two other metrics, Davies–Bouldin index (DB index) and Calinski-Harabasz index. For agglomerative clustering, both DB index and Calinski-Harabasz gave 5 and 6 as the optimal number of clusters with very small difference in the values. For K-Means and GMM clusterings, the DB index aligned with the previously obtained Silhouette index result of 3 and 2 clusters respectively. Calinski score for K-Means and GMM gave higher number of clusters and so we discarded it.

Thus, as per our empirical analysis agglomerative clustering works better than the other two algorithms. Our model primarily aims to aid organizations to find out the frequency of types of injections. This model can be run in a batch mode after a substantial amount of injections are encountered (for example, after every 5000 injections) or in a timed mode (for example, after every 6 months) as per organizations' convenience. However, one limitation with this approach is that some manual effort of mapping the clusters is still required. If we have a dataset wherein we know the type of SQL injections, we can build a more accurate model by using supervised machine learning algorithms.

5 Conclusion

In this work, we considered three unsupervised machine learning algorithms for clustering SQL injections. Our empirical study proved that the agglomerative clustering algorithm is better suited for our dataset. Since the data points are unlabelled w.r.t., the type of injections, some manual inspection is still required even after forming clusters to map the clusters to the specific types. This technique of clustering SQL injections can help organizations to figure out the frequency of occurrence of each type of attack. This would result in judicious deployment of resources to prevent SQL injection attacks. We understand that there is still scope for performing clustering using other advanced machine learning techniques which may give better results. In the coming future, we also plan to work on labelling SQL injections for the benefit of the research community.

References

1. Security statement by Freepik (2020). https://www.freepikcompany.com/newsroom/statement-on-security-incident-at-freepik-company/
2. SQL injection attack: A major application security threat (June 2020). https://www.kratikal.com/blog/sql-injection-attack-a-major-application-security-threat/
3. OWASP Top 10 2017 (2017). https://owasp.org/www-project-top-ten/2017/
4. SQL injections cyber attacks (September 2017). https://outpost24.com/blog/SQL-injections-cyberattacks

5. Zhang, K. (2019, November). A machine learning based approach to identify SQL injection vulnerabilities. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 1286-1288). IEEE.
6. Tripathy, D., Gohil, R., and Halabi, T. (2020, May). Detecting SQL injection attacks in cloud SaaS using machine learning. In 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS) (pp. 145-150). IEEE.
7. Farooq, U. (2021). Ensemble machine learning approaches for detection of sql injection attack. Tehnički glasnik, 15(1), 112-120.
8. Wang, Y., Wang, D., Zhao, W., & Liu, Y. (2015, July). Detecting SQL vulnerability attack based on the dynamic and static analysis technology.In 2015 IEEE 39th Annual Computer Software and Applications Conference (Vol. 3, pp. 604-607). IEEE.
9. Kindy, D. A., & Pathan, A. S. K. (2011, June). A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques. In 2011 IEEE 15th international symposium on consumer electronics (ISCE) (pp. 468-471). IEEE.
10. Halfond, W. G., Viegas, J., & Orso, A. (2006, March). A classification of SQL-injection attacks and countermeasures. In Proceedings of the IEEE international symposium on secure software engineering (Vol. 1, pp. 13-15). IEEE.
11. Kasim, Ö. (2021). An ensemble classification-based approach to detect attack level of SQL injections. Journal of Information Security and Applications, 59, 102852.
12. Courant, J. (2020, December). Developer-Proof Prevention of SQL Injections. In International Symposium on Foundations and Practice of Security (pp. 82-99). Springer, Cham.
13. Gandhi, N., Patel, J., Sisodiya, R., Doshi, N., & Mishra, S. (2021, March). A CNN-BiLSTM based Approach for Detection of SQL Injection Attacks. In 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE) (pp. 378-383). IEEE.
14. Falor, A., Hirani, M., Vedant, H., Mehta, P., & Krishnan, D. (2022). A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks. In Proceedings of Data Analytics and Management (pp. 293-304). Springer, Singapore.
15. Jemal, I., Cheikhrouhou, O., Hamam, H., & Mahfoudhi, A. (2020). SQL Injection Attack Detection and Prevention Techniques Using Machine Learning. In International Journal of Applied Engineering Research 15.6 (pp. 569-580).
16. "SQL injection dataset", 2021
https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset