Artificial Intelligence Techniques for SQL Injection Attack Detection

John Irungu
University of the District of Columbia
Washington, District of Columbia
john.irungu@udc.edu

Steffi Graham
University of the District of Columbia
Washington, District of Columbia
steffi.graham@udc.edu

Anteneh Girma
University of the District of Columbia
Washington, District of Columbia
anteneh.girma@udc.edu

Thabet Kacem
University of the District of Columbia
Washington, District of Columbia
thabet.kacem@udc.edu

ABSTRACT

In recent years, web-based platforms and business applications have been rising in popularity deeming themselves indispensable as they constitute the main backbone of business processes and information sharing. However, the unprecedented increased number of cyber-attacks have been threatening their day-to-day operations. In particular, the Standard Query Language Injection Attack (SQLIA) remains one of the most prevalent cyber attacks targeting web-based applications. As a consequence, the SQLIA detection techniques need to be constantly revamped and stay up-to-date in order to achieve the full potential of mitigating such threats. In this paper, we propose an artificial intelligence model based on supervised machine learning techniques to detect SQLIA. As part of the proposed model, we introduce an input string validation technique as a primary anomaly identifier using pattern matching for SQL Query data with anomalies-injections. To evaluate our approach we injected one type of SQLIA that is tautology attacks and measured the performance of our model. We used three main classifiers in our model and our findings indicate a model prediction accuracy of 98.3605% for Support Vector Machine (SVM), 96.296% for K-Nearest Neighbors (KNN), and 97.530% for Random Forest. The approach proposed in this paper has the potential of being used to integrate an automated SQL Injection detection mechanism with Intrusion Detection Systems (IDS) and Intrusion Protection Systems (IPS).

CCS CONCEPTS

• **Security and privacy → Artificial immune systems**.

KEYWORDS

Standard Query Language Injection, Machine Learning, Intrusion Detection System

ICIIT 2023, February 24–26, 2023, Da Nang, Vietnam

ACM ISBN 978-1-4503-9961-6/23/02…$15.00
https://doi.org/10.1145/3591569.3591576

ACM Reference Format:
John Irungu, Steffi Graham, Anteneh Girma, and Thabet Kacem. 2023. Artificial Intelligence Techniques for SQL Injection Attack Detection. In *2023 8th International Conference on Intelligent Information Technology (ICIIT 2023), February 24–26, 2023, Da Nang, Vietnam.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3591569.3591576

1 INTRODUCTION

Web-based applications are distributed systems with heterogeneous execution platforms such as web browsers, web servers, and network connections [3]. In recent years, these web-based applications have evolved to become indispensable technologies that facilitate business processes running efficiently while using different hardware platforms. To illustrate this growth, the web design industry market revenue was $11 billion [5] as of 2022. Also, the ease of use of web applications enabled the web traffic to increase by 53.74% of web visits on mobile phones and 46.26% on desktops [6].

Despite the attractive benefits of web applications in modern businesses, vulnerabilities in these web applications are ubiquitous and are likely to be exploited by attackers targeting web applications which account for 43% of hacking breaches. The Standard Query Language Injection Attack (SQLIA) is one of the most dangerous and most widespread cyber-attacks targeting web-based applications. SQLIA appends some input to a Standard Query Language (SQL) query to get an unauthorized access to a database then escalate the privilege before engaging into malicious activity that may span from data breach to integrity breach. SQLIA can also be described as a software vulnerability caused by the absence or proper input validation [9]. In fact, one of the most used SQL injection tactics by attackers is the manipulation of the username and password in SQL queries where an attacker can manipulate the script by adding malicious commands that bypass the authentication protocols.

In most web-based applications, authorization is mainly affected by the query language where the back-end database is manipulated. Attackers penetrate web applications to steal information and access the database by altering the query structure. This, in turn, authorizes access to a malicious attacker posing a threat to the application by corrupting its strings of command statements [16]. The prevalence of SQL injection attacks can be attributed to the industry's transition of most business processes from offline to online platforms. Since the web-based infrastructure tends to be more prone to attacks than the traditional offline systems and databases, the SQLIA mostly target web applications in relational

databases where the Internet traffic frequency is relatively high leading to a large attack surface. These attacks are generally carried out by adding unauthorized characters or commands in the SQL code thus enabling the attacker to execute a malicious query [17]. An example of web application SQL injection is the user login and password authentication where a more administrative user information can be bypassed in the query to enter additional 'user' inputs in order to capture the authentication details such as a username in a web-based accounts.

This paper investigates how the SQL query input validation can be leveraged to detect SQLIA through query string, length, and character positioning. Data labeling is subsequently implemented to identify the query outliers. The outliers are categorized as anomalies in the query pattern analysis and are thus detected and classified through machine learning classifiers including SVM, KNN and Random Forest to detect the SQL injection attacks. The artificial intelligence model is implemented through data streaming technologies that are useful in detecting SQL injection attacks in web-based applications. As far a the dataset used for the validation of this approach, it represents network log data.

The rest of the paper is organized as follows. Sections 2 discusses the literature review and related work. Section 3 provides an overview of the different types of SQL injection attacks. Section 4 describes the common prevention mechanisms against SQLI. The methodology and experimental results are described, respectively, in Section 5 and 6. Section 7 provides a detailed discussion about how our approach compares to the other techniques used to detect and present SQLIA. Section 8 concludes the paper and provides a glimpse of the future work.

2 RELATED WORK

According to Shreya et al. [11], sanitizing and filtrating inputs such as line breaks, and single quotes are one of the countermeasures against SQL injections. However, it is difficult to guarantee 100% protection. The authors also acknowledge the lack of input validation as the main cause of SQL injection and propose a BCRYPT hashing of the username and password and string-matching algorithm. The BCRYPT function hash is described as having a 'slow' conversion calculation to force the attacker to use more time and resources. The main drawback against hashing is the emergence of sophisticated powerful computing power that can crack the hash through brute-force capabilities which may also have an optimal time than the BCRYPT hash.

Deep learning has also been used to analyze the sanitization of the extracted features to detect SQL Injections. Zhang [23] proposed a Convolutional Neural Network (CNN) to detect SQL vulnerabilities based on features extracted from source code files achieving a precision score of 95.4%. Zhang also used Multi-Layer Perceptron (MLP) achieving a recall of 63.7%. Comparatively, deep learning has more resources in terms of computing power, unlike Machine Learning. Although Zhang's study of detecting SQL Injection based on the vulnerability data yielded high accuracy score, he acknowledges a low recall of 66% whose drawback may be the models failing to detect at least a third of the vulnerabilities.

Bronjon et al [7] proposed a Machine Learning (ML) and Natural Language Processing (NLP) to detect SQL Injections. The authors used programmatic techniques such as input filtering, validation, and parameterized queries to detect SQL injections. The traditional versus programmatic techniques comparative study affirms the capability of using ML and NLP techniques where Naïve Bayes, Decision Tree, Ada Boost, and Random Forest algorithms were used with an F-1 score of 98%. The experimental findings noted that linear and non-linear SVM scored higher than other classifiers that were used in their study. Our study achieves similar results to theirs.

Rubidha et al. [2] provided a comprehensive overview of the nature of SQL Injections and topics surrounding mitigation strategies. In their study, the authors argue that the tautology attacks are one of the most severe and damaging injection attacks to organizational data which occur by bypassing authentication. Our work agrees with their stand on the severity of tautology attacks and we further simulate it in our evaluation. Our study also concurs with the authors' findings that as more security measures are developed to prevent SQL Injections, more sophisticated attacks and vulnerabilities continue to emerge.

Muslihi et al. [12] compared 14 techniques to detect SQL Injections and ranked the code injection as one of the top-rated security vulnerabilities by the Open Web Application Security Project. The survey review by the authors reinforces the leverage and potential of machine learning and deep learning technologies in solving complex injection vulnerabilities. According to the authors, Long short-term memory (LSTM), CNN, MLP, Deep Belief network (DBN) and Bidirectional LSTM constitute the major techniques used by researchers. The survey evaluation, which relied on deep learning, found synonymous detection and performance evaluation methods, it also found existing gaps in the use of real-time network traffic detection. On the other hand, our paper proposes a mechanism to use real-time detection of network packets as an optimal and more effective way to not only detect SQL injections but also gain an analytical edge for future mitigation techniques. Also, our approach relies on Machine Learning data streaming for the detection of SQL injections as it is more resource friendly when compared with the deep learning approach that the authors proposed.

Based on the previous studies on SQLIA, it is agreeable by most of these findings that programmatic techniques for detecting SQLIA are gaining traction over traditional methods. Machine learning and deep learning are the main techniques used nowadays to develop intelligent models that can detect SQLIA and accurately analyze the results. In this study, we used artificial intelligence and input validation to enhance the real-time detection of SQLIA.

3 TYPES OF SQL INJECTIONS ATTACKS

There are various types of SQL injection attacks: error-based attacks, tautology attacks, and union-based attacks. These attacks are known for distorting and deleting databases through unauthorized query changes. Actually, both relational and non-relational databases, such as Cassandra and Mongo DB, [10] are vulnerabe to SQLIA.

3.1 Tautology Attacks

In this type of attack, attackers use tokens whose code evaluates as true which is in turn used to bypass the authentication process

by 'cunning' the database. These attacks are mainly executed by unauthorized use and by the inclusion of the "WHERE" conditional statement. The query statement is always true due to the inclusion of the tautology command. Figure 6a shows s sample authentic SQL query in black. The malicious query, shown in blue, has a 'where' clause "1=1" that always evaluates to true thus deeming the attack possible.

```
SELECT * FROM user WHERE name
                = "xyz" AND password = "345cba"

SELECT *  FROM user WHERE name
= xyz "AND password= "345cba" OR  "1" = "1"
```

Figure 1: Tautology Attack Pseudo -Code

3.2 Union-based Attack

In this is type of SQL injection attack, an invalid query is merged with an authentic one using the UNION statement [15]. This results into an unauthorized data access to another column being to the outcome of the authentic query, incurring more outcomes than required. A sample union-based attack is shown in Figure 2 where the malicious query adds the username and password column to the outcome of the legitimate query making the attacker gain access to this privileged information.

```
SELECT  m,n FROM table 4  (authentic)

SELECT m.n FROM table 4 UNION SELECT k.l

FROM table 5 (union base query)

UNION SELECT username,password FROM username
```

Figure 2: Union Attack Pseudo-Code

3.3 Error-based Injections

While union-based attacks combine multiple SQL queries to steal information through modification of the data [17], error-based attacks provide errors that indirectly lead them to acquire information about the structure of the database. A vulnerability that would be prone to error-based SQL injection attack is when the development phase of web applications is done on a live site instead of on an offline or a restricted platform. Unintentional vulnerabilities to error-based SQL injections may lead to unauthorized access to information and technical failures such as command injections. When error-based command injections occur, the developer may encounter error messages from the system such as buffer overruns, catching exceptions, and format strings.

3.4 Piggy-backed Injections

Piggy-backed attacks add malicious code to the queries by using delimiters that allow attackers to append additional queries to the original ones. The added queries may be used to distort the database or drop important information rendering the database inefficient and unreliable.

Figure 3 provides an example of such an attack where the query above the second query after ";" drops the table name resulting in a loss of information in the database.

```
SELECT * FROM user WHERE name
                = "xyz" AND password
                = "345cba"; DROP table name
```

Figure 3: Piggy-backed Attack Pseudo-Code

With billions of connected devices through the Internet of Things in more than 250 countries, the risks that are associated with information theft, loss, and misuse are directly proportional to the ease of information access and interoperability. This is due to the different security standards of manufacturers [20]. The connected devices and infrastructure rely on databases for the storage of information and records. The result of such attacks is information theft, Denial of Service Attacks (DDOS), technical failure, and incurred losses due to downtime.

4 PREVENTATIVE MECHANISMS

With the increasing use of cloud-based infrastructure the interoperability of web-based devices and applications, the threat of SQLI attacks will increase. Therefore, there is a need to mitigate SQLIA by adopting stealth measures that are intelligent, and reliable to counter sophisticated attacks. There are different ways that can be used to mitigate SQL injection attacks and enhance the integrity of the database systems. Some of the existing SQLIA prevention mechanisms are described as follows.

4.1 Deployment of IPS signatures

The primary means of information security is to ensure that information assets and systems that host them are safe from threats and vulnerabilities. To enhance this safety against SQLI Attacks, intrusion protection systems can be used to prevent unauthorized modification of the database and unauthorized access to the servers by attackers. This can be accomplished by enforcing the policy compliance to the deployment of IPS as a control measure against unauthorized access to the database systems.

An IPS is a network security tool that monitors the network for intrusions [[22] and alerts or blocks them. It can be hardware-based or software-based. For instance, Apache web server can detect and block attempts to access the database through exploits or weak vulnerabilities [1]. Although an IPS does not specifically control the SQLI query alterations, it can control the access to the query strings connected to the server. It also automates the intrusion detection process [19]. Besides, it can detect anomalous activities, vulnerabilities, and policy violations of application traffic in addition to alerting the IDS of any malicious activity in the network traffic. It also monitors the network traffic [13] for possible embedded ultra-modern attacks and works inline with the data streams to prevent attacks in real time.

4.2 Query Hashing Techniques

Hashing is one of the methods that ensure data integrity against intrusion from unauthorized subjects. Through hashing, an output called a hash is generated based on the input data, which in turn prevents a string of fixed characters from remaining static regardless of the length or size of the input. This results into big savings in term of space in the query repository and time thanks to the use of unique primary key in the hashing function. For instance, PHP version 5 and above has in-built 40 hash algorithms that generate hash keys with 8 to 128 characters [1]. Due to the high collision rate of the smaller key and large storage for large keys, 32-character keys are preferable for the hashing purpose. A hash is a mathematical operation on the data that makes it difficult to capture the password due to the addition of permutations that would be required to get the right password [19]. Although password hashing has been secure overtime, modern computers are able of carrying out millions of permutations in a second and this can make hashing vulnerable due to the implementation of brute force attacks by malicious attackers which creates a table of failed attempts and passwords.

5 METHODOLOGY

5.1 Approach

In this paper, we propose an artificial intelligence model that detects anomalies in SQL queries by introducing a machine learning algorithm to predict and annotate anomalies of SQL injections through query input validation. As part of the model, we developed a classification model using SVM, KNN, and Random Forest to detect the anomalies. Python-MySQL was used to connect the integrated a Python module that enables SQL integration and data manipulation in both environments as depicted in the algorithm 1.

Algorithm 1 depicts a Python Database Application Programming Inter-phase (API) connecting to a database engine. In step 1, we installed the Python SQL toolkit called SQLAlchemy. Steps 2 and 3 show a call to create an engine for the Python API connection at the local host port 3306. Once the engine is created, the subsequent steps 5 and 6 show the data interaction between SQL by calling python the methods.

Algorithm 1: Python SQL database module

```
1 #! pip install SQLAlchemy
2 from sqlalchemy create_engine
3 engine=create_engine(='mysql+pymysql://root:password
4 @127.0.0.1:3036/data')
5 df.to_sql("test_table", engine, if_exists='replace')
6 df=pd.read_sql('SELECT * FROM test_table;' ,engine)
7 df
```

After the SQL-Python engine is created, data manipulation and SQL Queries can be invoked in python. The anomalies from the SQL input validation pattern were labeled as '1' and '0'. The labeled data are then called in a python environment for data pre-processing and feature engineering respectively. The pseudo-code in Algorithm 2 describes the anomaly prediction process once the data has been fetched from SQL.

As shown in Algorithm 2, the fetching of the data with the labeled attributes '0' and '1' representing benign and malicious, respectively. After labeling is done, the data predictor and target variables are then split into X and y as shown in step 3. In step 4, the split variables were then trained and tested in an 80:20 ratio. We then applied SVM, KNN, and Random Forest as classifiers to get the prediction results.

Algorithm 2: ML-based SQL Injection Detection

1 **Step 1:** Load the data in a python environment with the attributes and errors;
2 **Step 2:** If not categorized, add an error column and categorize the SQL injection queries as "1" or "0";
3 **Step 3:** Split the data into X and Y X= Predictor Y= Target (anomaly);
4 **Step 4:** Train the data 80% of the data and set 20% for testing;
5 **Step 5:** Using SVM, KNN and Random Forest the model and test the accuracy ;
6 **Proposed accuracy tests:** SVM, KNN, Random Forest

Feature engineering was done using supervised learning in binary classification to predict the errors/anomalies in SQL queries.

5.2 SQL Input Validation

A key component in our approach is the SQL query input validation as it is the primary means of detecting SQL injection attacks. The hypothesis was tested through simulation of the Tautology Attack where a conditional clause is added to the query statement to pull data. As found in the two different queries, the legitimate query has a length 'x' whereas the tautology query has a length higher than x based on added characters in the query statement condition. This approach has the potential be applied to a web platform to flag queries that may have different character length from the standard database queries.

The rational we followed in the input validation relies on the following mathematical formulation for the length of the query statement and is based on the pattern of the statements where y_i is the i-th position of letter in pattern D, and the string of the query that follows certain pattern injections can be detected by the following process.

Let us consider the following sequences S = y_1,y_2 ... y_n. The query string can be represented by $y_1 y_2 \ldots y_T y_{T+1} \ldots y_n$ and the pattern can be represented by $x_1 x_2 \ldots x_T x_{T+1} \ldots x_n$. The variable x_i and u_i are random variables [4]. In this formulation, S is also a pattern denoting a sequence or pattern of characters from y_1 to y_n. In the case of our SQL query, this may represent the characters in the query. A standard query may have a list of the same characters of y_i in S. An injection occurs in the query where the number of characters S will increase. Subsequently, x_i, which represents a pattern of sequence characters y_1, y_2 ... y_T ... y_n, will increase leading to a distinctive length or size of the pattern. The term u_i represents the position of y_i in x_i. The difference in pattern size in the SQL queries are indicators of a change in query statement which may be a result of a conditional function such as a tautology

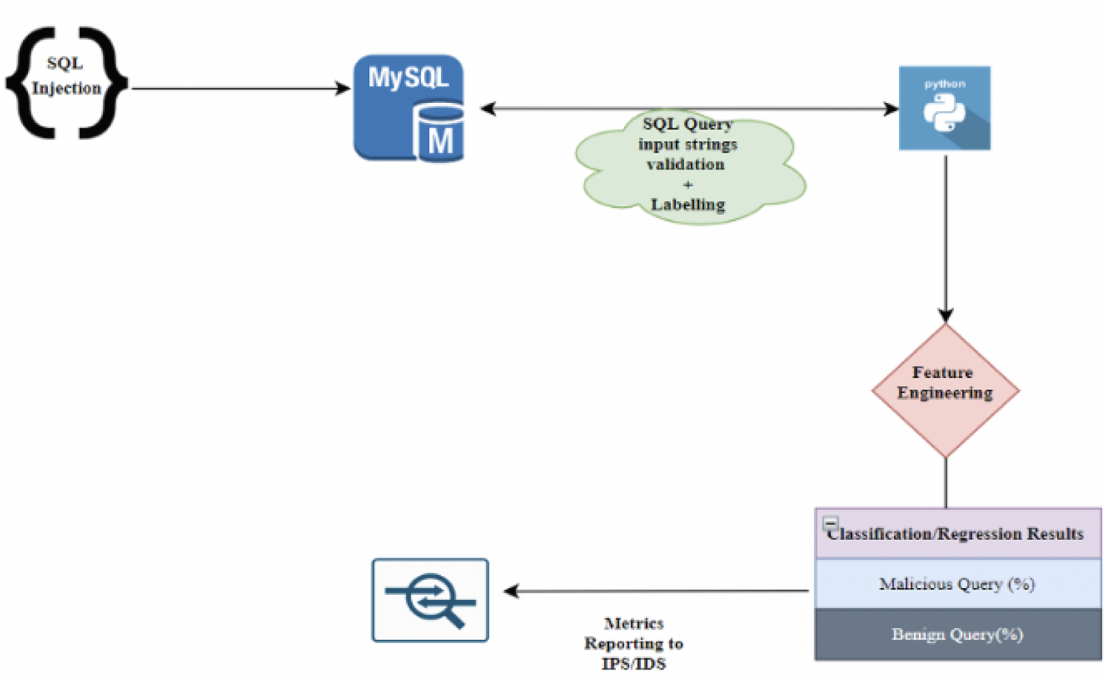


Figure 4: Architectural View of the Proposed Approach

attack. The difference in pattern size can then be used to label the query as malicious or not based on the uniformity of the pattern as the baseline.

By employing the above criteria, the position of unique characters in SQL query can be identified and the probability of its occurrence in the pattern can be calculated. Further, the size and position of the string can be estimated by verifying the pattern of string characters of the SQL queries from the position of characters in SQL statements.

The probability that X_T is a pattern of W is proportional to the product of the probability and length of w as shown in equation 1 where |w| is length of pattern w [4] while normalization factor NF is given by equation 2.

$$prob(X_T = w) = \frac{P_w.|w|}{NF} \tag{1}$$

$$NF = \sum_{all\,w} P_w.w \tag{2}$$

In such a scenario, where outliers appear in the SQL query length or characters positioning, the anomaly in the pattern is labeled as '1' or '0' and the data is exported for further analysis in the python-based feature engineering of the categorical labeled data.

Further, the collection of data from the queries is also necessary so as to distinguish outliers in the frequency of the query input by the users.

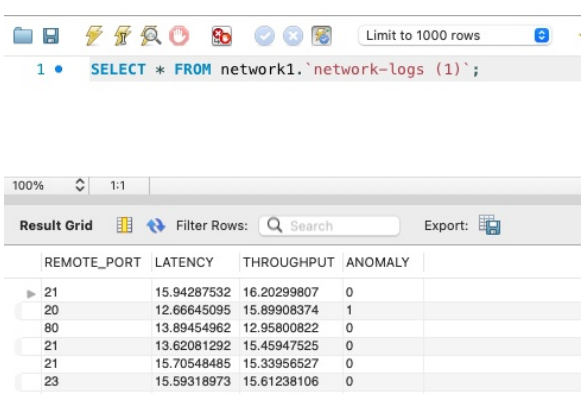
In an instance where the SQL queries within a specific time 'K' has the same data fetching patterns, supervised learning can be used to get the main features in the SQL pattern. Another key strength of our approach is that the pattern of SQL statements can further be integrated in data streaming technologies for real time alert to the IDS or IPS in case of prevalent malicious pattern disruptions that may be caused by an SQL injection attacks. An easy way to calculate the length of characters would be the use of the length function, due to the variance in authentication characters, to measure the length of the characters which may not be the most efficient way to detect anomalies. However, when the "where" conditional statements are added to the SQL query, our input validation is capable of the detecting anomalies through pattern recognition because of the additional change in character length and the presence of more data-types from alien query commands.

Through our Python-SQL database module, the audited data can be pre-processed for the prediction of attacks and other anomalies in the SQL queries and patterns. We used Support Vector Machine (SVM) supervised learning for the classification model due to its suitability in outlier detection. [14] ,and the instance-based learning of KNN and Random Forest which is more accurate without much hyper-parameter tuning.

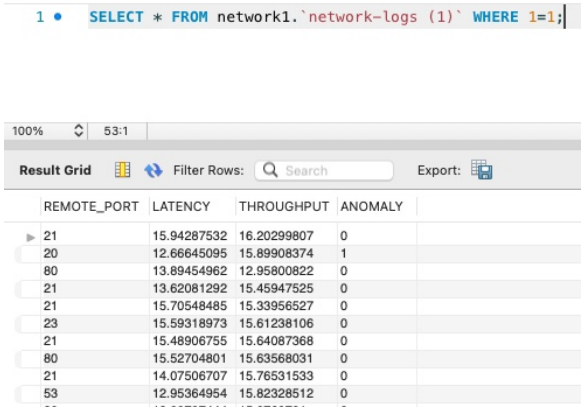
6 EVALUATION

6.1 Data Description

Data from the sensor network logs, in CSV format, were used for this experiment. The dataset was retrieved from a verified Pack-Publishing open dataset repository [13] and the network packets were analyzed in our simulation. The dataset had 405 rows and 3 columns of different features representing the remote port, latency, and throughput. The throughput measures the successful



(a) Normal Query



(b) Tautology Injection

Figure 5: Effect of Tautology Attack

data packets transferred within a time frame mainly in bits/second [8]. Latency is the time taken by the network packet to arrive from the source to the destination. We imported the dataset in CSV format in the SQL Workbench for query test analysis and we leveraged the SQLAlchemy connection to perform the simulation and obtain the experimental results.

The network logs files were used to simulate the tautology attack in our paper and to show how the input string validation can thwart SQLIA while the classifiers show the accuracy of properly classifying an attack. The data was imported into an SQL database where several functions were run to simulate a tautology attack and our input string validation module. The size and position of the SQL statement string were used to categorize the functions in different columns as benign and malicious for the simulated tautology attack functions which had a different input string in the SQL queries. We focused our simulation on a tautology SQL injection and a piggy backed SQL injection which results to wiping out of data using the drop command.

Figure 5a shows the legitimate query that fetches the data from the network1 table through the "select" function. The output of the query is the whole data from all columns in the network1 table.

The depicted tautology injection attack, shown in figure 5b, bypasses the "WHERE" clause by using the 'OR 1=1' condition that always evaluates to true. The 'OR 1=1' injection condition, results to the output of all the data stored and this may lead to the display of unauthorized data. Tautology attack may therefore be used to steal information by bypassing the authentication mechanisms.

A sample piggy back attack, as shown in figure 6, can be maliciously used to delete and completely wipe out enterprise data resulting to the loss of vital data.

In figure 6a, we pulled a request which selected data from the network1 table while in figure 6b we simulate a piggy back attack the 'DROP' query drops the whole 'network table' erasing all the data as seen below.

The data queries with multiple functions are the baseline for pattern matching where the SQL Injections queries and legitimate queries vary in anomaly characteristics that were used in the simulation. After labeling of the anomaly was done using the input validation, the data from SQL is connected to the Python environment for analysis to predict the accuracy of the validation technique.

For the purpose of this study, we simulated and labeled 10 tautology SQL injection attacks. This constitutes only 2.469% of total possible benign and malicious SQL queries. To predict the accuracy in python, the malicious injection was labeled as the response variable y and from the predictor X. Using the sklearn module in Python, we split the data in 70:30 fashion for training and testing, respectively.The experimental results are shown in Section 6.

6.2 Experimental Results

Based on our experiments, we evaluated the simulation using three classifiers namely KNN, SVM and Random Forest. We also measured the performance evaluation of the classifiers using different benchmark metrics including the precision, the recall and the F-1 score. The mathematical formulations to derive these metrics are described next.

The precision measures how accurate and precise the results are by measuring the threshold of true positives from the predicted positives [21]. It is derived by equation 3:

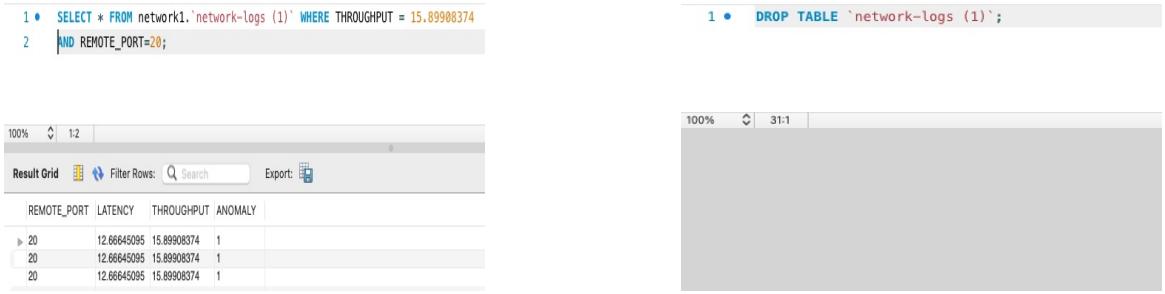
$$\frac{TP}{TP + FP} \tag{3}$$

The F-1 Score is a function of precision and recall and which seeks a balance between them. It is given by equation 4:

$$\frac{TP}{TP + \frac{1}{2}.(FP + FN)} \tag{4}$$

The recall measures the sensitivity of the actual positives compared to all observations including False Negatives and True Positives, as shown in question 5:

$$\frac{TP}{TP + FN} \tag{5}$$



(a) Normal Query **(b) Piggy Back Attack**

Figure 6: Effect of Piggy Back Attack

Table 1 , table 2 and table 3 show the result we obtained after running the model with the SVM, KNN and Random Forest classifiers.

Table 1: SVM Classification Results

	Precision	Sensitivity	F1-Score	Support
Benign	0.99	1.00	0.99	79
Malicious	1.00	0.55	0.71	2
macro avg	0.99	0.75	0.83	81
weighted avg	0.99	0.99	0.99	81
accuracy			0.98	81

Table 2: KNN Classification Results

	Precision	Sensitivity	F1-Score	Support
Benign	0.96	1.00	0.98	77
Malicious	1.00	0.25	0.40	4
macro avg	0.98	0.62	0.69	81
weighted avg	0.96	0.96	0.95	81
accuracy			0.96	81

Table 3: Random Forest Classification Results

	Precision	Sensitivity	F1-Score	Support
Benign	0.97	1.00	0.99	73
Malicious	1.00	0.33	0.50	8
macro avg	0.99	0.67	0.74	81
weighted avg	0.98	0.98	0.97	81
accuracy			0.97	81

As derived from the simulation results from the three classifiers, the SVM classifier had the highest prediction at a near 99% accuracy, followed by Random Forest at 97.53% and KNN at 96.3% prediction of anomalies-injections. In the binary classification, normal data queries with no anomalies were labeled as benign while data with detected anomalies injection was labeled as malicious.

The precision for the three classifiers had a high confidence score with the least being KNN with a 96% precision. The high precision metric shows that our model had a low false positive rate which is good for our anomaly prediction model.

7 DISCUSSION

In comparison to our approach, the most noticeable techniques that can be used to prevent SQLIA include installing up-to-date security patches to the databases and web servers from vendors, and utilizing the least privilege policy in provisioning accounts linked to the SQL database. Even though our study agrees with some of these techniques, it avoids most of their flaws as the security patch software from vendors is susceptible to even more sophisticated cyber supply chain malware attacks. This is considered as an emerging tactics, techniques and procedures (TTPs) embedded in the software updates and fraudulent SSL certificates that target vendors and clients to get initial access. In addition, the least privilege practices may not prevent human errors which contribute to a substantial amount of cyber vulnerabilities. This study, therefore, proposes an artificial intelligence-based approach as the most effective preventive and deterrence strategy against SQL Injection Attacks since it is scalable to the size of the features and pattern prediction and even deployment. Actually, our Python-SQL module is capable of fetching labeled malicious queries, which has the potential for being implemented in big data technologies using data streaming and also using deep learning techniques. Also, unlike supervised learning, deep learning has more pattern recognition and computing power.

8 CONCLUSION AND FUTURE WORK

In recent years, SQLIA caused enormous losses in the industry as a result of data loss, sabotage, and disruption of business processes by attackers. It is estimated that 65% government and commerce suffer from minor to major data breaches yearly [18] due to SQL injections attacks year. Therefore, the need for proper SQLIA mitigation should be done through the deployment of defense mechanisms that are more intelligent for sophisticated attacks, reliable, and cost-effective.

With the constant increase of IoT-based technologies, the industry will still face increased SQL injection attacks targeting web-based applications that may lead to detrimental vulnerabilities such

as information theft, distortion, and distributed denial of services. These attacks lead to the loss of millions of dollars to companies and even a total collapse of the supply chain [18]. To mitigate these threats, artificial intelligence can play a big role in threat detection and alerting of malicious attacks such as SQLIA.

In this paper, we proposed a machine learning model for detecting and thwarting SQLIA that relies on input string validation that enables the model to distinguish legitimate queries from injections based on the pattern recognition of the string and the size of the query. Further in order to automate the validation and identification of malicious SQLIA, we proposed the use of a Python-SQL database pipeline where malicious SQL queries were labeled in a binary categorization ('0' or '1', 'M' or 'B'). Through supervised learning, we predicted the queries that had a malicious SQLIA and did not pass the input validation step.

In future, we intend to further test and deploy Artificial Neural Networks (ANN) in to enhance SQLIA detection. Based on our findings, we expect to achieve a better performance in detecting SQLIA. To automate and enhance the collection of data from SQL databases and prediction of SQLIA, we intend to test data streaming technologies for optimal performance.

REFERENCES

[1] Cisco Community. [n. d.]. IPS Detects SQL Injection over https. ([n. d.]). https://community.cisco.com/t5/network-security/ips-detects-sql-injection-over-https/td-p/1670002

[2] Rubidha Devi, R. Venkatesan, and Raghuraman Koteeswaran. 2016. A study on SQL injection techniques. *International Journal of Pharmacy and Technology* 8, 22405–22415.

[3] Giuseppe A Di Lucca and Anna Rita Fasolino. 2006. Testing Web-based applications: The state of the art and future trends. *Information and Software Technology* 48, 12 (2006), 1172–1186.

[4] Jian Feng, Daniel Q Naiman, and Bret Cooper. 2007. Probability-based pattern recognition and statistical framework for randomization: modeling tandem mass spectrum/peptide sequence false match frequencies. *Bioinformatics* 23, 17 (2007), 2210–2217.

[5] J. Flynn. [n. d.]. 25+ WONDERFUL WEBSITE DESIGN STATISTICS. ([n. d.]). https://www.zippia.com/advice/website-design-statistics/

[6] Statcounter GlobalStats. [n. d.]. What percentage of Internet traffic is mobile? ([n. d.]).

[7] Bronjon Gogoi, Tasiruddin Ahmed, and Arabinda Dutta. 2021. Defending against SQL Injection Attacks in Web Applications using Machine Learning and Natural Language Processing. In *2021 IEEE 18th India Council International Conference (INDICON)*. IEEE, 1–6.

[8] P.R. Hicks. 1988. Throughput and delay performance measurement of packet switched data networks. In *IEEE Global Telecommunications Conference and Exhibition. Communications for the Information Age*. 527–534 vol.1. https://doi.org/10.1109/GLOCOM.1988.25897

[9] Zar Chi Su Su Hlaing and Myo Khaing. 2020. A detection and prevention technique on sql injection attacks. In *2020 IEEE Conference on Computer Applications (ICCA)*. IEEE, 1–6.

[10] KR Jothi, Nishant Pandey, Pradyumn Beriwal, Abhinandan Amarajan, et al. 2021. An efficient SQL injection detection system using deep learning. In *2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*. IEEE, 442–445.

[11] Shreya Kini, Anushka Parvate Patil, M Pooja, and Adithya Balasubramanyam. 2022. SQL Injection Detection and Prevention using Aho-Corasick Pattern Matching Algorithm. In *2022 3rd International Conference for Emerging Technology (INCET)*. IEEE, 1–6.

[12] Muhammad Takdir Muslihi and Daniyal Alghazzawi. 2020. Detecting SQL Injection On Web Application Using Deep Learning Techniques: A Systematic Literature Review. In *2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE)*. 1–6. https://doi.org/10.1109/ICVEE50212.2020.9243198

[13] Alessandro Parisi. 2019. *Hands-On Machine Learning for Cybersecurity*. https://www.packtpub.com/in/big-data-and-business-intelligence/hands-machine-learning-cybersecurity?utm_source=github&utm_medium=repository&utm_campaign=9781788992282

[14] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[15] PortSwigger. [n. d.]. SQL injection UNION attacks. ([n. d.]).

[16] M Amutha Prabakar, M KarthiKeyan, and K Marimuthu. [n. d.]. An efficient technique for preventing SQL injection attack using pattern matching algorithm.

[17] Saima Saleem, Muhammad Sheeraz, Muhammad Hanif, and Umar Farooq. 2020. Web Server Attack Detection using Machine Learning. In *2020 International Conference on Cyber Warfare and Security (ICCWS)*. IEEE, 1–7.

[18] David E. Sanger. [n. d.]. Pipeline attacks yields urgent message about US cybersecurity. ([n. d.]). https://www.nytimes.com/2021/05/14/us/politics/pipeline-hack.html

[19] Karen Scarfone, Peter Mell, et al. 2007. Guide to intrusion detection and prevention systems (idps). *NIST special publication* 800, 2007 (2007), 94.

[20] Yash Shah and Shamik Sengupta. 2020. A survey on Classification of Cyber-attacks on IoT and IIoT devices. In *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 0406–0413.

[21] Koo Ping Shung. 2018. Accuracy, precision, recall or F1. *Towards data science* 15, 03 (2018).

[22] VMWare. 2022. What is an intrusion prevention system? (2022).

[23] Kevin Zhang. 2019. A machine learning based approach to identify SQL injection vulnerabilities. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1286–1288.