Using Data Mining Techniques for the Detection of SQL Injection Attacks on Database Systems

Añasco, Cesar [1]; Morocho, Karen[1]; Hallo, María[1]

[1]*Escuela Politécnica Nacional, Facultad de Ingeniería en Sistemas, Quito, Ecuador*

Abstract: In any business organization, database infrastructures are subject to various structured query language (SQL) injection attacks, such as tautologies, alternative coding, stored procedures, use of the union operator, piggyback, among others. This article describes a data mining project developed to mitigate the problem of identifying SQL injection attacks on databases. The project was conducted using an adaptation of the cross-industry standard process for data mining (CRISP-DM) methodology. A total of 12 python libraries was used for cleaning, transformation, and modeling. The anomaly detection model was carried out using clustering by the k – nearest neighbors (kNN) algorithm. The query text was analyzed for the groups with anomalies to identify sentences presenting attack traces. A web interface was implemented to display the daily summary of the attacks found. The information source was obtained from the transactions log of a PostgreSQL database server. Our results allowed the identification of different attacks by injection of SQL code above 80%. The execution time for processing half a million transaction log was approximately 60 minutes using a computer with the following characteristics: Intel® Core i7 processor 7th generation, 12GB RAM and 500GB SSD.

Keywords: Log, Database Attacks, Anomalies, Queries, CRISP-DM, IDS (Intrusion Detection Systems)

Uso de Técnicas de Minería de Datos para la Detección de Ataques de Inyección de SQL en Sistemas de Bases de Datos

Resumen: En cualquier organización empresarial, las infraestructuras de bases de datos y de almacenamiento de la información están sujetas a diversos ataques de inyección de lenguaje de consulta estructurado (SQL), tales como: tautologías, codificación alternativa, procedimientos almacenados, uso del operador unión, consultas adicionales, entre otros. Este artículo describe un proyecto de minería de datos para desarrollar una herramienta que identifique ataques a bases de datos por inyección de código SQL. El proyecto se realizó con una adaptación de la metodología de proceso estándar de la industria para la minería de datos (CRISP-DM). En el desarrollo, se usó un total de 12 librerías de Python para la limpieza, transformación y modelado. El modelo de detección de anomalías fue realizado usando agrupación mediante el algoritmo de vecinos más cercanos (kNN), y a los grupos con anomalías se realizó el análisis del texto de la consulta para identificar sentencias que indiquen un indicio de ataque. Para la visualización de los resultados de los modelos, se implementó una interfaz web, la cual despliega la estadística diaria de los ataques encontrados. La fuente de información se obtuvo de registros de transacciones del log de un servidor de base de datos PostgreSQL. El resultado obtenido permitió la identificación de diferentes ataques por inyección de código SQL por encima del 80% y el tiempo de ejecución para el procesamiento de medio millón de registros fue de aproximadamente 60 minutos, mediante una computadora con las siguientes características: procesador Intel® Core i7 de séptima generación, 12GB de RAM y disco sólido SSD de 500GB.

Palabras clave: Log, Ataques a Bases de Datos, Anomalías, Consultas SQL, CRISP-DM, IDS (Sistema de Detección de Intrusos)

1. INTRODUCTION

In any business organization, business database and storage infrastructures contain primarily sensitive data, which are targets of a wide range of attacks on their security by users seeking to take advantage of that information for different purposes (Malik y Patel, 2016). The most common attacks are due to the wide range of vulnerabilities of databases. They are mainly due to the lack of visibility when they occur at the database level and to the scarce analysis of vulnerabilities. An example is the inadequate management of user rights and the

*cesar.anasco@epn.edu.ec
Recibido: 03/12/2021
Aceptado: 30/09/2022
Publicado en línea: 01/05/2023
10.33333/rp.vol51n2.02
CC 4.0

lack of monitoring for unauthorized access (Telefónica Company, 2015).

Modern security vulnerability detection applications must be reliable, functional, and easy to manage. In recent years, intrusion detection systems (IDS) based on data mining have demonstrated high precision, good generalization of new types of intrusion and robust behavior in a changing environment (Chetan y Ashoka, 2012). In addition, the security incident response and intrusion identification engines are gaining relevance given the growth of malware and various threats that databases are exposed to.

Research in intrusion detection methods relies on statistical models. Lee et al. (2000) analyzed log regular expressions from SQL queries to identify the normal behavior of users. Wang et.al (2020) proposed an automatic neighbor selection based on the kNN (k-Nearest Neighbors) algorithm. Making clear emphasis on showing that supervised learning methods have a higher efficiency than unsupervised methods and explaining the advantages of using a machine learning algorithm, such as kNN, for anomalies detection with massive log files. Brahma and Panigrini (2020) analyzed outlier values using data mining techniques to automate the intrusion detection process with greater precision, using data generated from database users. We consider these mentioned studies to be the primary source of our project development, we combined a hybrid approach of SQL regular expressions analysis for identifying specific types of injections and outlier detection to decrease the complexity of working with large datasets.

This paper presents the results of a project to implement a model which identifies SQL code injection attacks through the analysis of various PostgreSQL transaction logs using the kNN data mining technique to detect atypical queries. This solution was based on the observation that the transaction record of an injection attack has a considerably different duration time than a typical transaction and generally other types of SQL instructions. The authors also developed a plain-text analysis function to identify each type of SQL injection attack. The results showed that the algorithm correctly identifies most tested SQL injections.

2. BACKGROUND

a. SQL Code Injection

SQL code Injection attack is defined as an attack that is carried out on a web application. The attacker enters SQL code in an input box (web form) to obtain illegal access to the application (Kumar y Pateriya, 2012).

b. Types of SQL Code Injection

- Tautologies: In this type of SQL injection, the attacker can inject malicious code into the query based on the conditional "or 1 = 1" that will always be evaluated as true (Al-Sayid y Aldlaeen, 2013).
- Logically Incorrect Queries: The attacker takes advantage of the invalid queries that were run with an error message to obtain information about the tables or data types (Varshney y Ujjwal, 2019).
- Inference Boolean injection: The attacker draws his logical conclusions from a true-false question thrown into the database (Varshney y Ujjwal, 2019).
- Inference Time-based: It is an inferential SQL injection technique that is based on sending an SQL query to the database, forcing the database to wait a specific amount of time (in seconds) before responding (Varshney y Ujjwal, 2019).
- Union Query: The attacker adds malicious code to a second secure query using the "union" operator to obtain information from the table of interest in the database (Varshney y Ujjwal, 2019).
- Piggy-Backed Queries: In this type of SQL injection, the attacker manipulates the data using the clauses: INSERT, UPDATE and DELETE without altering the logic of the original query (Al-Sayid y Aldlaeen, 2013).
- Stored Procedure: The attacker integrates malicious SQL injection codes into existing stored procedures in databases (Varshney y Ujjwal, 2019).
- Alternate Encodings: The attacker uses different encodings such as hexadecimal, ASCII, or Unicode to confuse the SQL statements avoiding basic validation (Al-Sayid y Aldlaeen, 2013).

c. Types of anomaly detection techniques

Table 1 compares the different data mining techniques to detect anomalies, their advantages, disadvantages, and the type of algorithm each technique follows.

Table 1. Comparison of the algorithms to identify anomalies

Name	Algorithm Type	Supervised Yes/No	Advantages	Disadvantages
kNN	Classification algorithm	Yes	Simple algorithm to train. Make assumptions about the distribution that the data follows are unnecessary.	Slow if you are working with a large amount of data.
Isolation Forest	Based on proximity	No	Efficient algorithm for outlier detection, especially in high-dimensional datasets.	Some of these model complexity increases with the data dimension and size. The training data is not labeled; the model must learn without a teacher.
CBLOF	Based on proximity	No	It has optimal performance with a small size of dimensions. Fast execution with large data sets,	It is not capable of capturing anomalies for a high-dimensional data set.
HBOS	Based on proximity	No	It performs well on global anomaly detection problems.	It cannot detect local outliers.
Feature Bagging	Outlier set	No	When training a data set, you take a sample of random characteristics rather than the entirety.	It should be combined with other methods to improve the predictive accuracy of outliers.

3. METHODOLOGY

In this study, the CRISP-DM methodology was adapted and applied to identify, select, process, and analyze the development of our artifact, which is the tool for intrusion detection. The steps that were carried out are the following:

a. Problem Understanding: It is crucial to understand the objectives and requirements of the study.

b. Data Understanding: Collecting quality data that brings us closer to the problem. Data is then scanned and verified, and unnecessary data is discarded.

c. Data preparation: Data is prepared, cleaned, and integrated into an appropriate format for the data mining technique used later.

d. Modeling: This phase focuses on selecting the appropriate modeling technique for the data mining project that is being developed.

e. Model Evaluation: Evaluate that the steps of the model are carried out correctly and measure that each result obtained is following the objectives.

f. Visualization: The final phase allows observing the system prototype data mining model.

3.1. Problem Understanding

This study aims to develop a system prototype that allows the identification and prediction of events of SQL injection attacks on PostgreSQL database servers.
An extensive bibliographic search was carried out in files, documents, books, among other references.

We needed reliable and accurate information, we consulted scientific studies from virtual academic libraries such as Research Gate, Google Scholar, among others, to know the advances in this field. Also, in this step, the programming language we will use is analyzed. Finally, it is verified that it has the necessary libraries to process large amounts of data to solve the problem.

The literature reviewed suggested that the authors should focus on increasing its accuracy and efficiency at the network or operating system level (Hu y Panda, 2004). Other documents of interest described approaches of different authors that brought us closer to understanding the problem. Lee et al. (2000) proposed a database intrusion detection method that looked at frequent temporary data access.

Muslihi y Alghazzawi (2020) proposed a probabilistic SQL injection detection technique through deep learning techniques. Brahma y Panigrahi (2020) used fuzzy logic in the intrusion detection behavior model in the database to identify intrusion on PostgreSQL databases. All of the above research is based on statistical methods. But data mining techniques are showing their potential in database-based intrusion detection.

3.2. Data Understanding

The data of interest in this project are plain text files containing sets of transaction logs generated daily by the CSIRT (Computer Security Incident Response Center) PostgreSQL server. The log transactions generated in PostgreSQL are written daily in a formatted file (.log) to be transformed into new files (.csv). For this, a VMWare Workstation 16.1 virtual machine with Windows 10 Operating System was used, and PostgreSQL version 9.5 was installed, which is the database management system of interest in this project. The first test data we worked on was 62 Megabytes of the transaction log.

The list below describes each unique value in the PostgreSQL configuration file for transactions log that simulates normal behavior.

- **% d:** Name of the database that is being used to run the queries.
- **% u:** Name of the user who connected to PostgreSQL server.
- **% r:** Host and remote port. The client making the connection must be on the same machine as the PostgreSQL server.
- **% p:** ID of the process. Unique identifier assigned to each transactions log.
- **% m:** Timestamp. It is the timestamp that includes milliseconds.
- **% s:** Is the transactions log on timestamp.
- **% i:** The command label returns the statement, such as: SELECT, INSERT, UPDATE, DELETE.

3.3. Data preparation

In this stage, the input data is processed so that it can be used in the chosen data mining technique, in this case kNN.

The steps we follow for data processing are detailed below:

3.3.1. Data Collection

Table 2 lists all the data selected and created from reading the log transaction set. The transaction log is collected in a text file format(.log) that is initially unstructured, so we read the file line by line to extract the data from each log using the libraries: pandas, CSV and Python as a programming language.

The files generated from this reading process are a document (.csv) with the attributes described in Table 2 and a separate text document where the LOG_ID and the RESULT of each set of log transactions are stored for later analysis of the type of attack in the evaluation stage.

This step is crucial because it is unnecessary to use the sort technique to have a long string of text, such as the query statement for each transaction record. With the new CSV file, the PyOD tool allows us to preview the transaction log in the Cartesian plane according to the chosen variables without needing to make prior changes to the normalization of the values to observe them graphically. This phase helps to understand the behavior of the transaction log to have a clear view of the best variables that can be used to identify anomalies.

Table 2. Description of each attribute of log

Attribute Name	Description	Domain	Type
LOG_ID	Unique identifier generated for each transactions log	discrete	integer
BDD	Name of the database on which the transactions log was generated	categorical	string
USER	Name of the user who connected to PostgreSQL	categorical	string
IP	IP address of each computer	integer	integer
DATE	Date of the day the query was made to obtain the transactions log.	continuous	date
HOUR	Time of the query with milliseconds	continuous	time
DATE AND HOUR	Date and time of the day on which the query was made to obtain the log.	continuous	date
COMMAND	Type of sentence executed registered in the log.	categorical	string
DURATION	Duration time in seconds that the query takes to execute.	continuous	double
RESULT	Query execution result	categorical	string

Figure 1a shows the number of log transactions generated inside the database per hour. Based on this graphic representation, we can see that there are certain hours of the day when the queries made to the server are less active. Figure 1b shows a different plane in which the type of query performed by the log transactions in the database is displayed. It can be seen in detail that throughout the day, the most unusual commands are UPDATE, GRANT, ALTER, COPY.

In Figure 2, generated by the pyplot library, we can see a high density of transactions log generated in each query to the database for each hour of the day. It is necessary to know the dimension of the data to be processed and how to use it efficiently (Brahma y Panigrahi, 2020), that is why we present the density of queries done within the hours as shown in Figures 2a, 2b and 2c. These graphs helped to understand what type of values are most important in the generated log transactions and use them in the mining model, which will appropriately identify a behavior different from the normal one (Charania y Vyas, 2016).

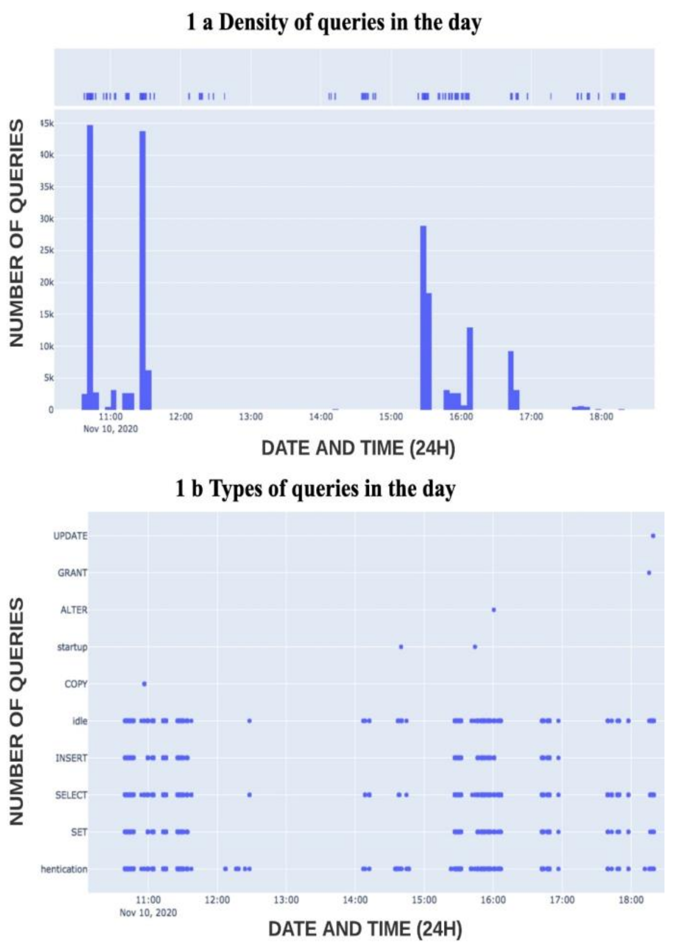


Figure 1. Number of consultations per hour (1a) bar graph and (1b) types graph

3.3.2. Data Cleaning and Normalization

Certain log transactions collected with null values on the query date and time were eliminated so as not to interfere with the identification of anomalies.

Once the unnecessary columns and null values in the log transactions have been eliminated, taking into account the data analysis carried out with the PyOD tool using the kNN technique, we decide to generate separate files of 5 000 log transactions in each file. With each file generated in the previous steps, we normalize them. This means reshaping their data space so that their distribution along the Cartesian plane dimensions is approximately the same.

The traditional normalization method used for kNN is minimum-maximum normalization. Minimum-maximum normalization subtracts the minimum value of an attribute from each attribute value and divides the difference by the attribute range. These new values are multiplied by the new range of the attribute to get the time of execution and the time of the day and finally added to the new minimum value (Cho, 2002). These operations transform the data into a new range [0,1]. As can be seen in the formula;

$$Xnew = \frac{X - Min(x)}{Max(X) - Min(X)} \tag{1}$$

Where:

Xnew: Normalized Value

Min(x): Minimum value of the characteristic

Max(x): Maximum value of the characteristic

3.4. Modeling

Data analysis is the third stage, where the model is applied to detect SQL injection attacks. The steps followed are these:

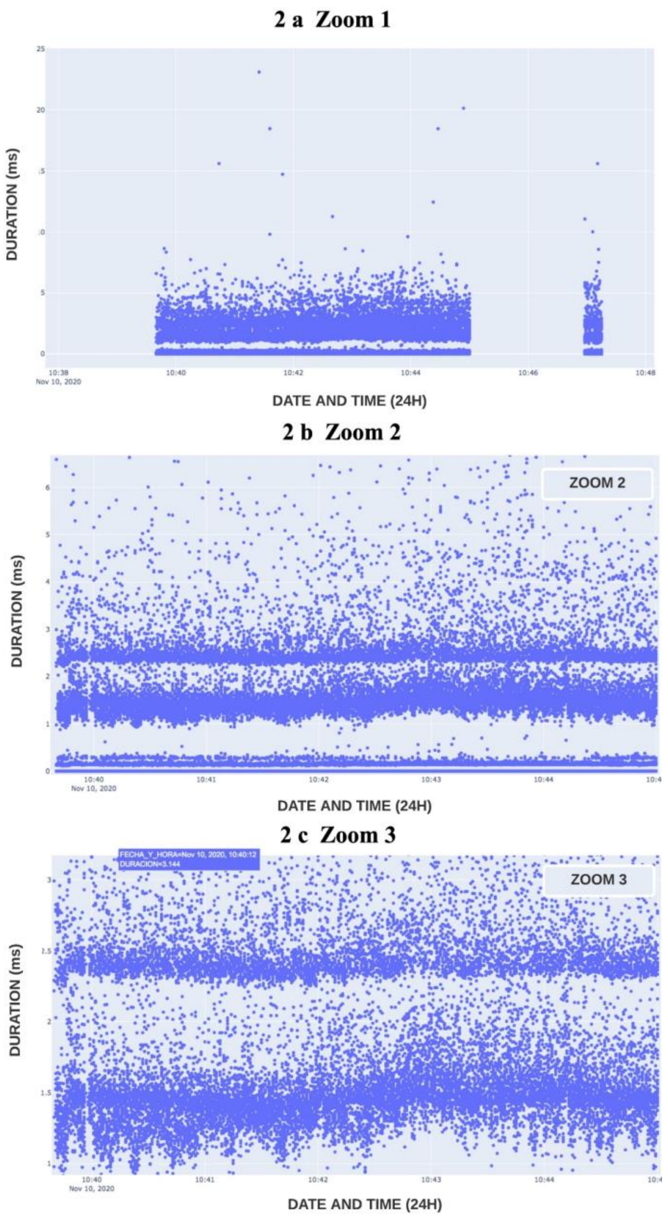


Figure 2. Log transactions duration and query time in the day (2a) Zoom1, (2b) Zoom 2 and (2c) Zoom 3

3.4.1 Anomalies Identification on Prepared Data

To identify anomalies (outliers) within a data set we use the PyOD library. This process is performed after the log transaction set has passed the data transformation and cleansing process. For this identification, a comparative evaluation between several algorithms was first carried out to understand how kNN works and why this algorithm was the best option in this project. This comparison is listed in Table 1 and supported in the introduction. kNN was used within a plane comparing the query duration milliseconds and the query time in day. The duration of the executed query is the most critical factor in identifying database server injection attacks (Gong et al., 2019).

kNN Algorithm

The kNN classifier is based on a distance function that measures the difference or similarity between two instances (Wang et al., 2020). The standard Euclidean distance "d (x, y)" between two instances "x" and "y" is defined by the formula:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad (2)$$

Where:

x: i-th highlight of the instance

y: i-th highlight of the instance and is the total number of features in the data set

It must be considered that the kNN algorithm is a non-parametric prediction method. This means that it does not require the elaboration of a previous model. To make a prediction, the kNN algorithm does not compute a predictive model from a training data set as in linear or logistic regression. Therefore, for kNN, there is no actual learning phase. This is why it is generally classified as a lazy learning method (Wang et al., 2020).

Figure 3 shows the implementation of the kNN (k-Nearest Neighbor) algorithm used for the atypical queries identification.

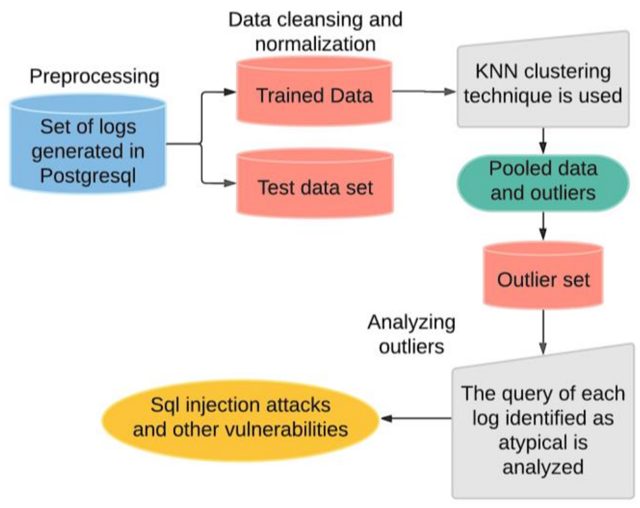

Preprocessing
Set of logs generated in Postgresql
Data cleansing and normalization
Trained Data
Test data set
KNN clustering technique is used
Pooled data and outliers
Outlier set
Analyzing outliers
The query of each log identified as atypical is analyzed
Sql injection attacks and other vulnerabilities

Figure 3. kNN Model for detection of atypical queries

The result of the aggrupation and anomalies identification is shown in Figure 4. The figure shows X and 'Y-axis normalized values. The black dots represent the anomalies detected by kNN, and the dots inside the red line, or colored white, represent the regular transactions log.

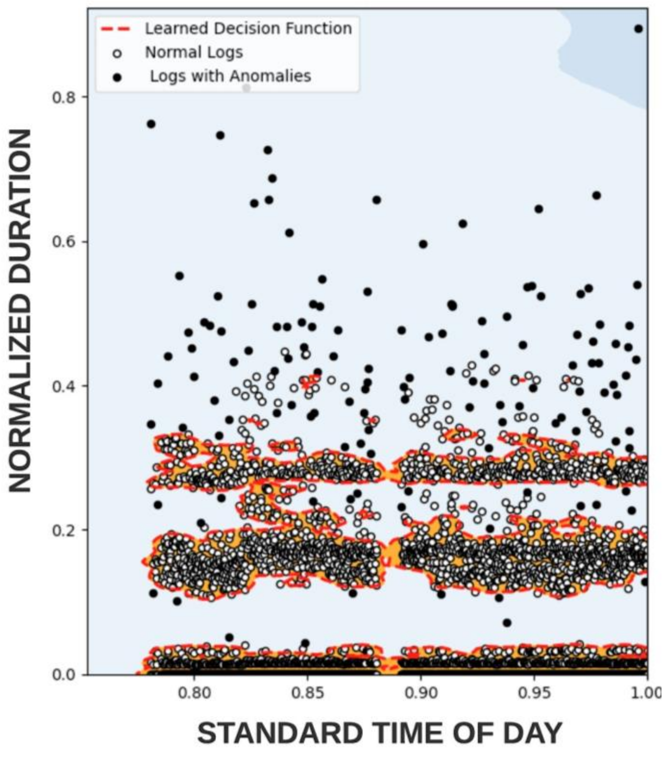


Figure 4. Anomalies Identified with kNN algorithm

3.4.2. Queries Text Analysis

The transactions log identified by the implemented algorithm as anomalies are stored in a new file. In this file, we found which anomalies are eventually a malicious attack, using text analysis on each query string to identify statements that are different from regular queries.

The following list shows the expressions used on the URL of the database server to simulate SQL injection attacks. Each query was tested with the model to identify the expressions on the SQL query.

- Tautology: http://localhost:3434/users.php?name=gabriela and 1=1--+
- Logically Incorrect Queries: http://localhost:3434/users.php?name=gabriela and select * from gestion.ges_catalogo where cat_nombre = 'belleza'latina'
- Union Query: http://localhost:3434/users.php?name=gabriela and union select prov_ruc, prov_razon_social, 22 from activo. proveedor where '1'='1' --+
- Piggy-Backed Queries: http://localhost:3434/bodegas.php?name=BODEGA1; drop table gestion.ges_usuario --+
- Inference Boolean injection: http://localhost:3434/bodegas.php?name=bodega1 and declare @s varchar (8000); select @s = prueba (); if (ascii (substring (@s, 1, 1)) & (power (2, 0))) > 0 waitfor delay '0:0:5'
- Inference Time-based: http://localhost:3434/users.php?name=gabriela pg_sleep(15) --+
- Stored Procedure: http://localhost:3434/users.php?name=gabriela and select *from gestion.ges_usuario where usr_nombre = '' and usr_password = ''; drop table gestion.ges_usuario --+
- Alternate Encodings: http://localhost:3434/users.php?name=legalUser? exec(CHAR(0x73687574646f776e)) -- AND usr_password='' --+

These listed SQL injection attacks have unique text expressions, and in this way, the authors were able to identify what type of SQL injection it is.

3.5. Model Evaluation

After establishing a classification model with defined test data, the next step is to determine how effective the implemented algorithm is in correctly identifying the transactions log that leaves traces of an SQL injection. For this, 50 SQL injection attacks were simulated on our PostgreSQL-based server. The result was obtaining 100 log transactions, 50 having SQL injections registered in the queries of each log transaction, and 50 log transactions with 'normal' behavior without any hint of attack. The results were as listed below in Table 3:

Table 3. Classification model performance results

	Prediction			
Model Results	**Attack Transactions Log**		**Normal Transactions Log**	
Positive	**TP**	84%	**FP**	10%
Negative	**FN**	16%	**TN**	90%

$$Accuracy\colon \frac{TP+TN}{TP+FP+FN+TN} * 100\% = 87\% \quad (3)$$

$$Precision\colon \frac{TP}{TP+FP} * 100\% = 89.36\% \quad (4)$$

$$Sensibility\colon \frac{TP}{TP+TN} * 100\% = 84\% \quad (5)$$

Where:

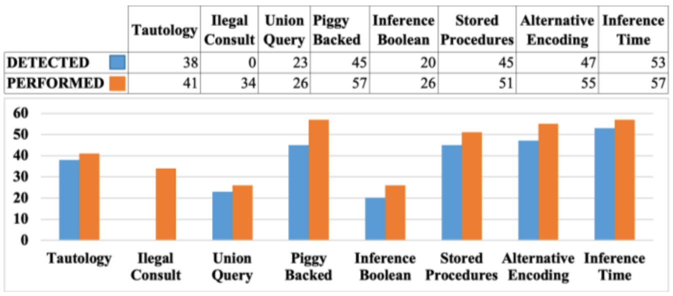
TP: True Positive	*TN:* True Negative
FN: False Negative	*FP:* False Positive

The evaluation values reflected that:

- Out of 50 known attacks in the log transaction set, 42 were identified by our model. While eight were discarded as normal behavior. Although the model has desirable accuracy, it could leave multiple attacks unidentified in a large data set.
- Percentage results for accuracy, precision, and sensitivity are nearly even. So, we can reiterate that the implemented model works.

3.5.1. Text query analysis evaluation

This section presents the results of the SQL code injection attack tests of the developed model. In Figure 5, we can see a summary of the number of attacks carried out by each type of injection and the number identified by the system prototype. The graph shows that in various injection types, the model performance is effective for this project**.**



	Tautology	Ilegal Consult	Union Query	Piggy Backed	Inference Boolean	Stored Procedures	Alternative Encoding	Inference Time
DETECTED 🟦	38	0	23	45	20	45	47	53
PERFORMED 🟧	41	34	26	57	26	51	55	57

Figure 5. Comparison between attacks made and detected

We can see that the percentage of identification of each type of injection exceeds 76% of cases except for the type of injection of "Illegal Queries", the main reason is that the attacks are based on the error response, and there is no specific SQL command that can be detected (Chaki y Mat, 2019).

These percentages agree with the results of the performance calculation obtained from the implementation of the kNN based classification algorithm shown on the model evaluation. Although the obtained training set can achieve a good balance in size and precision, there is still some noise data, which will affect anomaly detection accuracy to a certain degree. If the precision of the automatically selected training set can be improved, the accuracy of the anomaly detection model will be further improved.

3.5.2. Load testing

The following four steps were the evaluated stages of the CRISP-DM implementation with the log files.

1. Data Collection

This step collects the data from the entered file, transactions log reading, attributes extraction, and generation of new files. These files will contain the data of the extracted transactions log in a format established for the next step.

2. Data Separation

This step performs a division log, where files are generated with a maximum of 5 000 transaction sets each.

3. Data transformation and Algorithm Implementation

Each of the transaction sets generated in the previous step is captured, cleaned, and normalized for implementing the kNN algorithm. This step results in the generation of new files with the transactions identified as anomalies.

4. Text Analysis

A text analysis of the query made to the database associated with the log transactions identified as an anomaly is performed. In case of registering any expression that reveals the existence of an attack or its attempt, the anomaly registry is sent to the system server.

This evaluation consists in measuring the execution time of each step explained by loading different sizes of log transaction files. The measurement of the execution time of the files was carried out by using the 'time' command, which shows the following results:

- **USER (U):** The amount of CPU time spent on user-mode code (outside the kernel) within the process. This is just the actual CPU time used to run the process.
- **CPU (C):** Percentage of CPU that was allocated to the command.
- **TOTAL:** Sum of the three attributes indicated at the end. The result is shown in seconds.

The behavior when executing each step with a different number of log transactions is shown in Table 4. The data collection and separation steps have an almost linear behavior for execution time, so it is inferred that the size of log transactions used on them does not affect their performance. Unlike the two last steps, execution times take much longer when the transactions log sets are larger.

The data transformation and kNN Implementation step seems to have more complexity and delay time, as shown in the first three log files. However, in the rest of files specified in Table 4 we can see that the text analysis step has an exponential delay growth when the set of log transactions is more extensive. Although there is an increase in the delay time of the third step, where a delay of 21 minutes was obtained with approximately half-million log transactions, this is understandable since the identification algorithm is implemented there. Unlike the text analysis, whose uncontrolled growth is due to the registry function that is sent to the server every time an anomaly is found. Also, we can see that the step that makes the most use of the central processing unit (CPU) is the third step, due to the separation of sets of logs and the identification of anomalies carried out in this step. The other steps have a uniform and normal CPU usage for the tasks they perform.

3.6. Visualization

Once the model evaluation phase was finished, a web interface was created to visualize the model results. Figure 6 shows the tools used and the design of the solution architecture we used to develop the system prototype.

3.6.1. Architecture

Our overall solution architecture has the main components:

a. Data Source

The PostgreSQL source database data where the transaction log files are created.

b. Data Processing

All changes made to the original files are made here.

c. Modeling

The anomalies detection and text analysis are executed here.

d. Data Storage

We make use of a free software provider for web hosting applications.

e. Visualization

Finally, data is presented in graphs for the final users.

Table 4. Classification model performance results

FILES	TIME (s)													
SIZE (mb)	**2,6**		**5,2**		**10,4**		**20,7**		**41,7**		**83**		**166**	
LOGS	**10 000**		**20 000**		**40 000**		**80 000**		**160 000**		**320 000**		**640 000**	
VARIABLES	**U**	**C**	**U**	**C**	**U**	**C**	**U**	**C**	**U**	**C**	**U**	**C**	**U**	**C**
DATA COLLECTION	1	94%	1	82%	2	81%	4	76%	7	89%	16	84%	30	89%
Total	1,001		1,830		2,915		5,511		9,406		20,968		39,143	
DATA SEPARATION	1	75%	1	78%	2	65%	2	77%	2	73%	2	83%	3	64%
Total	2,352		2,326		3,097		2,650		3,247		3,581		7,058	
DATA TRANSFORMATION AND KNN IMPLEMENTATION	12	92%	23	96%	48	97%	87	97%	178	98%	365	98%	725	97%
Total	13,587		29,249		73,516		99,700		92,010		373,910		756,030	
TEXT ANALYSIS	4	33%	10	41%	36	50%	128	65%	465	77%	1784	85%	7164	91%
Total	12,460		23,433		59,900		199,430		507,220		2102,290		7870,950	
TOTAL (s)	29,400		56,838		139,428		307,291		611,883		2500,749		8673,181	

3.6.2. Interfaces

The following interfaces were created for the final visualization step:

a. Login

A login interface page was created for the web system. It contains the primary login fields and a button to retrieve the user's password if needed. The interface aimed to establish a layer of security for the users using the developed model.

b. Daily Summary Tab

This interface allows you to see daily summary graphs. The detected attacks graph displays the anomalies identified as attacks. These graphs show the attacks by hours and a division of the types of attacks.

c. Attack and Alert Log Tab

An interface was also created to show a table where the transactions log of all the attacks identified since the application start of operation are displayed. In the same way, it allows one to visualize in real time the attacks being detected and the associated data.

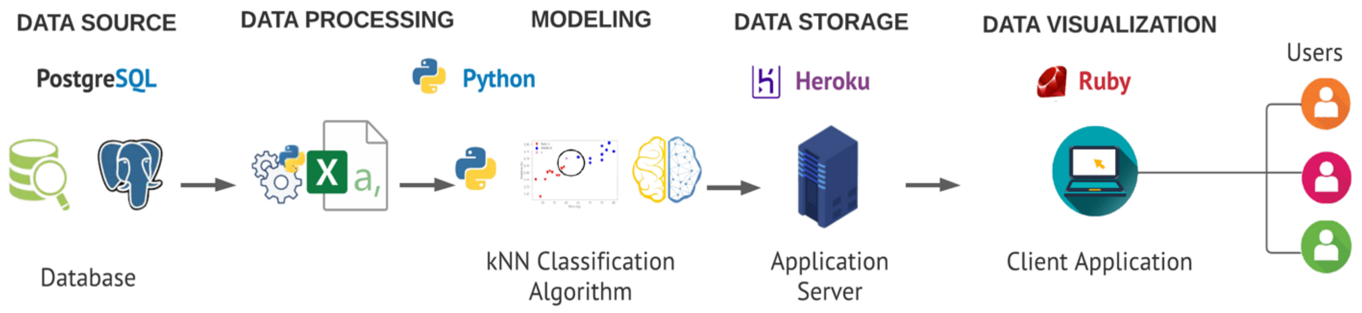


Figure 6. Solution Architecture

4. SOURCE CODE AVAILABILITY

Access to datasets and source code for reproduction are available from the corresponding author upon request**.**

5. DISCUSSION

Data preparation phase was executed with considerable work on data cleaning; we removed a significant amount of duplicate log transaction records because of empty columns, special characters, useless data, among others. As a result, a new set of data ready to be processed was obtained.

The kNN model developed was the most appropriate approach for our planned solution. When evaluating this model, we obtained an accuracy of 89.36%, which is reflected in equation 4 and it is a good indicator that the model meets its objective of detecting SQL injections.

It is important to mention that most of the data mining algorithms are less efficient when their data set increases, and so was the case in this project, it is recommended to use Big Data tools for a future approach on this scope.

6. CONCLUSIONS AND FUTURE WORK

In this study, we use the kNN classification algorithm because it is a simple algorithm that classifies the new data based on a measure of similarity. Additionally, it does not make assumptions about the distribution followed by the data and tells us that the closer a piece of data is to the data set has a normal behavior, the further away it is considered an anomaly.

The functionality tests in the evaluation stage reflected that the algorithm implemented in the system is practical and has the option to be implemented not only in PostgreSQL databases but also in others that can generate similar data used in this project.

From the project carried out, it is proposed to analyze the integration of the algorithm implemented for the identification of anomalies with a neural network since it has been shown in several current studies that a more robust and accurate intruder identifier can be obtained through the use of a constant learning network.

The kNN (k-Nearest Neighbor) algorithm correctly performed its objective and was detected SQL injection attacks when working with sets of 5 000 log transactions. Being a supervised learning algorithm, its implementation was straightforward, and its results were reflected by correctly classifying normal from abnormal behavior.

Big Data concepts can be added to improve the execution time of the application, one example is the use of parallelism with Hadoop Distributed File System (HDFS). This framework can reliably store process large amounts of data using simple programming models on a cluster. Furthermore, by distributing storage and computing across many servers, the resource can grow with demand while remaining inexpensive across all sizes (Shvachko et al., 2010).

It is recommended to investigate depth methods of identifying other types of attacks such as excess of privileges, denial of services (DoS), sensitive data without handling. There is a wide range of internal and external attacks, to which database servers can be vulnerable, different from those identified in this project.

REFERENCES

Al-Sayid, N. A., & Aldlaeen, D. (2013, March). Database security threats: A survey study. In 2013 5th International Conference on Computer Science and Information Technology (pp. 60-64). IEEE. https://pdf.zlibcdn.com/dtoken/a8e5836392db05a43b7d71fd52c2ffdd/CSIT.2013.6588759.pdf

Brahma, A., & Panigrahi, S. (2020). Role of soft outlier analysis in database intrusion detection. In Advanced Computing and Intelligent Engineering (pp. 479-489). Springer.

Chaki, S. M. H., & Din, M. M. (2019). A Survey on SQL Injection Prevention Methods. International Journal of Innovative Computing, 9(1). https://ijic.utm.my/index.php/ijic/article/view/224/143

Charania, S., & Vyas, V. (2016). SQL Injection Attack: Detection and Prevention. Int. Res. J. Eng. Technol, 2395-56. https://docplayer.net/49705074-Sql-injection-attack-detection-and-prevention.html

Chetan, R., & Ashoka, D. V. (2012, January). Data mining based network intrusion detection system: A database centric approach. In 2012 International Conference on Computer Communication and Informatics (pp. 1-6). IEEE. https://pdf.zlibcdn.com/dtoken/97d1ad3853bc5d09653697d9eab6958f/iccci.2012.6158816.pdf

Cho, S. B. (2002). Incorporating soft computing techniques into a probabilistic intrusion detection system. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), *32*(2), 154-160.

Gong, X., Zhou, Y., Bi, Y., He, M., Sheng, S., Qiu, H., ... & Lu, J. (2019, June). Estimating web attack detection via model uncertainty from inaccurate annotation. In 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom) (pp. 53-58). IEEE. https://pdf.zlibcdn.com/dtoken/a8e4f9aa33faa580677d87cbb2fe7c2f/CSCloud/EdgeCom.2019.00019.pdf

Hu, Y., & Panda, B. (2004, March). A data mining approach for database intrusion detection. In Proceedings of the 2004 ACM symposium on Applied computing (pp. 711-716).

Kumar, P., & Pateriya, R. K. (2012, July). A survey on SQL injection attacks, detection and prevention techniques. In 2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12) (pp. 1-5). IEEE.

Lee, V. C., Stankovic, J. A., & Son, S. H. (2000, May). Intrusion detection in real-time database systems via time signatures. In Proceedings Sixth IEEE Real-Time Technology and Applications Symposium. RTAS 2000 (pp. 124-133). IEEE.

Malik, M., & Patel, T. (2016). Database securityattacks and control methods. International Journal of Information, 6(1/2), 175-183. https://www.aircconline.com/ijist/V6N2/6216ijist18.pdf

Muslihi, M. T., & Alghazzawi, D. (2020, October). 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) (pp. 1-6). IEEE.

Shvachko, K., Kuang, H., Radia, S. y Chansler, R. (2010, mayo). El sistema de archivos distribuido Hadoop. En 2010, vigésimo sexto simposio de IEEE sobre sistemas y tecnologías de almacenamiento masivo (MSST) (pp. 1-10). IEEE.

Teléfonica Company. (2015). Bases de datos y sus vulnerabilidades más comunes. https://www.acens.com/wp-content/images/2015/03/vulnerabilidades-bbdd-wp-acens.pdf

Varshney, K., & Ujjwal, R. L. (2019). LsSQLIDP: Literature survey on SQL injection detection and prevention techniques. Journal of Statistics and Management Systems, *22*(2), 257-269

Wang, B., Ying, S., & Yang, Z. (2020). A Log-Based Anomaly Detection Method with Efficient Neighbor Searching and Automatic K Neighbor Selection. Scientific Programming, 2020. https://www.hindawi.com/journals/sp/2020/4365356

BIOGRAPHY

Cesar, Añasco was born in Quito - Ecuador in 1995. Graduated of the Computer and Computer Systems Engineering career (2021) at the National Polytechnic School. With professional experience in the area of Information Security. Research interests: Data Analysis and Web Development.

Karen, Morocho was born in Quito - Ecuador on January 31, 1996. Graduate of the Systems Engineering career at the National Polytechnic School. She has professional experience in the area of ICTs and technological solutions. Research interests: Web development, Databases, Big Data.

PhD. María Hallo is a professor at the Faculty of Systems Engineering of the National Polytechnic School, Quito-Ecuador. MSc in Computer Science Notre Dame de la Paix University, Namur, Belgium. PhD in Computer Science applications, University of Alicante. Research interests: Databases, Business Intelligence, Semantic Web, Digital Libraries.