Journal of Internet Services and Applications, 2024, 15:1, doi: 10.5753/jisa.2024.3799

Combining Regular Expressions and Machine Learning for SQL Injection Detection in Urban Computing

Michael S. Souza [**Universidade Estadual do Ceará (UECE)** | ***michael.souza@aluno.uece.br***]
Silvio E. S. B. Ribeiro [**Universidade Estadual do Ceará (UECE)** | ***silvio.eduardo@aluno.uece.br***]
Vanessa C. Lima [**Universidade Estadual do Ceará (UECE)** | ***vane.carvalho@aluno.uece.br***]
Francisco J. Cardoso [**Universidade Estadual do Ceará (UECE)** | ***fco.cardoso@aluno.uece.br***]
Rafael L. Gomes [**Universidade Estadual do Ceará (UECE)** | ***rafa.lopes@uece.br***]

Center of Science and Technology, State University of Ceará, Av. Silas Munguba 1700, Fortaleza, CE, 60714-903, Brazil.

Received: 31 October 2023 • **Accepted:** 01 May 2024 • **Published:** 02 July 2024

Abstract Given the vast amount of data generated in urban environments the rapid advancements in information technology urban environments and the continual advancements in information technology, several online urban services have emerged in recent years. These services employ relational databases to store the collected data, thereby making them vulnerable to potential threats, including SQL Injection (SQLi) attacks. Hence, there is a demand for security solutions that improve detection efficiency and satisfy the response time and scalability requirements of this detection process. Based on this existing demand, this article proposes an SQLi detection solution that combines Regular Expressions (RegEx) and Machine Learning (ML), called Two Layer approach of SQLi Detection (2LD-SQLi). The RegEx acts as a first layer of filtering for protection against SQLi inputs, improving the response time of 2LD-SQLi through RegEx filtering. From this filtering, it is analyzed by an ML model to detect SQLi, increasing the accuracy. Experiments, using a real dataset, suggest that 2LD-SQLi is suitable for detecting SQLi while meeting the efficiency and scalability issues.

Keywords: Security, Injection, Machine Learning, Regex.

1 Introduction

Recently, the expansion of cities and urban areas has led to numerous practical and managerial difficulties. Urban Computing has come into play to offer innovative solutions and help tackle these issues prevalent in large cities and metropolitan areas [Silveira *et al.*, 2023]. This approach works by rolling out services that utilize information from an enormous array of diverse data gathered in big cities, coming from various sources [Rodrigues *et al.*, 2019; Silveira *et al.*, 2023]. The data is sourced through various remote sensing methods, management strategies, and analytical models [Musznicki *et al.*, 2022; Silveira *et al.*, 2023].

With the extensive adoption of computing services, there arises a need for security, as devices and users become targets for various threats and harmful systems [Portela *et al.*, 2024; da Silva *et al.*, 2020]. These threats seek to exploit potential weak spots in the communication and data access processes. One notable vulnerability is insecure information access by harmful users and programs, which try to overcome urban computing services via SQL Injection attacks (SQLi) [Das *et al.*, 2019]. These attacks take advantage of security gaps in inputs (such as text fields and queries), tricking the system into processing the requests and exposing often sensitive data from authenticated clients and devices [Parashar *et al.*, 2021; Moreira *et al.*, 2021]. Nowadays, according to OWASP, SQLi is the top security threat to online services and applications on the Internet [Tang *et al.*, 2020].

At present, there is a variety of security solutions both in literature and in the industrial sector aimed at detecting SQLi in online services and/or the databases they utilize. However, these solutions often prioritize efficiency (accuracy) in identifying SQLi attacks, overlooking the effect of these security measures on the performance of urban computing services. For instance, resource-heavy Artificial Intelligence techniques, mainly Machine Learning (ML), have been used for SQLi detection [Xie *et al.*, 2019; Li *et al.*, 2019; Parashar *et al.*, 2021; Tang *et al.*, 2020; Portela *et al.*, 2023], but they have a response time considered lengthy (taking milliseconds for a single analysis). This delay directly influences service performance, as many services have significant constraints and must address scalability aspects due to the vast amount of data and communication involved [Musznicki *et al.*, 2022; Gomes *et al.*, 2020; Geldenhuys *et al.*, 2021; Lv *et al.*, 2020]. Hence, there is a need for security solutions that, besides being efficient in detecting SQLi and potential threats, also meet the criteria of processing and response time. Thus, a suitable approach is to analyze not all the inputs of the services with ML techniques, but only the inputs considered possible threats.

Within this context, this article introduces a solution composed of (*Regular Expressions* - RegEx) and Machine Learning, called Two Layer approach of SQLi Detection (2LD-SQLi). The RegEx serves as an initial filtering step to defend against SQLi threats, and to meet response time and scalability requirements. Next, the inputs considered threats by the RegEx are deeply analyzed by an ML model to detect SQLi cases. In general, the proposal aims to determine if a specific query possesses a structure that suggests an SQLi attempt, meaning it aims to filter these potential threats to prevent the unnecessary execution of ML detection (which has a high response time and consumes a lot of computing resources). Additionally, the 2LD-SQLi adjusts dynamically the order of the RegEx applied in the filtering process, aiming to speed up it. Thus, the proposed solution focuses on the balance between the efficiency of detecting SQLi and the response time of the solution to bring scalability to the cybersecurity process as a whole.

2LD-SQLi applies several strategies: (I) API to serve as an SQLi detection service in the Edge environment to support cybersecurity strategies of urban computing; (II) Split of the solution in both Edge and Cloud environments, focusing on the fast and scalable execution of the detection in the Edge (closer to the urban computing services), while the training of ML techniques and management of RegEx set, that are complex and high computing tasks, are performed in the Cloud meaning (high amount of computational resources and high availability to access these resources); and, (III) Adaptability of the detection process through the update of the set of RegEx (the structure of the RegEx and the order of testing), as well as the ML technique used together. In general, this article has the following contributions: (i) Development of a scalable solution for SQLi detection based on RegEx and ML; (ii) Design of architecture considering Edge and Cloud environment with the possibility to be integrated with other existing cybersecurity solutions through APIs; (iii) Proposition of an algorithm to dynamically adjust the ordering of RegEx analysis; and, (iv) Deep evaluation with experiments considering real SQLi dataset available in the literature and other evaluation metrics.

The outcomes of the experiments, utilizing a real dataset, indicate that the proposed solution possesses suitable SQLi threat detection efficiency while offering a response time that satisfies the demands of urban computing services. Various scenarios were assessed, testing the amount of RegEx in the database and the requests of SQLi detection process. Additionally, the results are contrasted with existing machine learning methods, with the aim of drawing a comparison with other solutions.

In our previous work [Souza *et al.*, 2023], we proposed a set of RegEx to be used as an analyzer of input data to identify possible SQLi attacks. However, this initial work did not focus on the detection of SQLi with high accuracy using modern techniques, such as ML. In this way, the current proposal has the following innovation issues in front of the previous work: (A) Integration of the RegEx approach with ML techniques to perform SQLi detection; and, (B) Evolution of RegExs in the set, focusing on new patterns of SQLi identified.

The proposed solution was developed within the framework of the Research and Development project between UECE and LACNIC[1], which seeks to develop industrial solutions for safeguarding sensitive data on the Internet, and detecting SQLi is one of the aspects considered in the protection process, as it compromises the privacy of urban computing service users.

The remainder of this paper is organized as: Section 2 presents existing solutions related to SQLi in general systems and in the context of urban computing, while Section 3 describes the proposed SQLi detection solution. Section 4 discusses the results of the experiments conducted and Section 5 concludes the article.

2 Related Work

This section describes the main works related to SQLi detection in the context of urban environments and general systems, that were published recently, considering performance and quality issues. Table 1 presents several existing proposals (column *Reference*), where the *Context* column represents the scenario/context where the proposal is used, while the *Focus* column highlights the strategy of the corresponding proposal.

Parashar *et al.* [2021] proposed a method to detect SQL injection (SQLi) in Information Technology Applications. They achieve this goal through the use of text summarization, which allows for data processing regardless of input size. By creating a supervised machine learning model from the summarization, they are able to automate SQLi classification. However, the author's approach does not focus on scalability issues, which may lead to longer response times and reduce the applicability of the solution in urban computing scenarios.

Xie *et al.* [2019] developed a method for detecting SQL injection attacks using Elastic-Pooling Convolutional Neural Networks (EP-CNNs). This method creates a bidirectional matrix that captures all the data without truncation and can effectively detect SQL injection attacks by identifying irregular correspondences in the data. Tang *et al.* [2020] describe an SQL injection detection method based on neural networks that aim to achieve high accuracy rates, around 99%. The authors conducted statistical research on default data and SQL injection data derived from URL access logs of internet providers. Based on the statistical results, the authors identified eight types of resources and trained a Multi-Layer Perceptron (MLP) model.

Li *et al.* [2019] propose a deep forest model for detecting complex SQL injection (SQLi) attacks. The model is based on the AdaBoost algorithm and concatenates the raw vector of characteristics and the average of the previous outputs as input to each layer. The error rate is used to update the weights in each layer, with resources assigned varying weights based on their influence on the results during different training processes. The proposed models are efficient in SQLi detection but come with high computational costs, resulting in processing times that are longer than ideal for Urban Computing. In a similar vein, Fadolalkarim *et al.* [2020] present AD-PROM, an anomaly detection system designed to protect relational databases against malicious agents. AD-PROM traces system calls executed by programs running on the computational system hosting the target database. Based on this tracing, AD-PROM analyses the control and data flows of the programs and constructs a Hidden Markov Model (HMM) that detects anomalies indicative of SQL in-

[1]https://programafrida.net/en/archivos/project/data-protection-system-based-on-anonymization-techniques-and-in-accordance-with-privacy-laws

Table 1. Related Work

Reference	Focus	Benefit	Limitation
Parashar *et al.* [2021]	Detect SQLi in IT Application	It applies text processing with ML	It disconsiders response time that affects scalability
Xie *et al.* [2019]	Usage of CNN for SQLi detection	High accuracy in SQLi detection	The detection time is high
Tang *et al.* [2020]	Utilization of ANN for SQLi detection	High accuracy in SQLi detection	The solution is limited to URL analysis
Li *et al.* [2019]	Deep forest model for detecting SQLi	High accuracy in SQLi detection	The solution demands a high computational cost
Fadolalkarim *et al.* [2020]	SQLi Detection in local databases	High accuracy in SQLi detection	The solution is limited to local databases
Crespo-Martínez *et al.* [2023]	Detection of SQLi based on LR	The choice of ML is based on an extensive evaluation of ML models	The solution focus only on protocol information of inflow data
Our Proposal	SQLi detection based on Regex and ML	Edge and Cloud approach with Dynamic update of RegEx order analysis to Increase Scalability	The set of RegEx is not evolved automatically.

jection occurrences on the database host.

Crespo-Martínez *et al.* [2023] demonstrate that it is possible to detect SQL injection attacks inflow data from protocol information through two datasets based on data from several SQLi attacks. After that, the authors evaluated several machine learning techniques to detect SQLi situations, where Logistic Regression presented the best results for the network flow data collected.

From these existing solutions, it is possible to note that no existing solution has the goal to develop a solution to detect SQLi considering scalability and response time issues, which is the focus of the proposed 2LD-SQLi (flexible and adaptable RegEx set integrated with ML model).

3 Two Layer approach of SQLi Detection (2LD-SQLi)

This section details the proposed 2LD-SQLi, where Section 3.1 describes the 2LD-SQLi designed architecture (discussing the defined modules and their communication flow) and Section 3.2 presents the background about SQLi, the set of RegEx defined and the dynamic order strategy applied. Finally, Section 3.3 show the ML techniques applied in the proposal.

3.1 Architecture of 2LD-SQLi

The 2LD-SQLi solution is composed of several modules: API, Filtering, Detection, Tranning, RegEx DB, and Update. Moreover, we consider the following types of communication: internal communication that occurs between the modules inside the same environment and Internet communication that is performed remotely through the API (using a secure channel). An overview of the 2LD-SQLi and deployment context is illustrated in Figure 1.

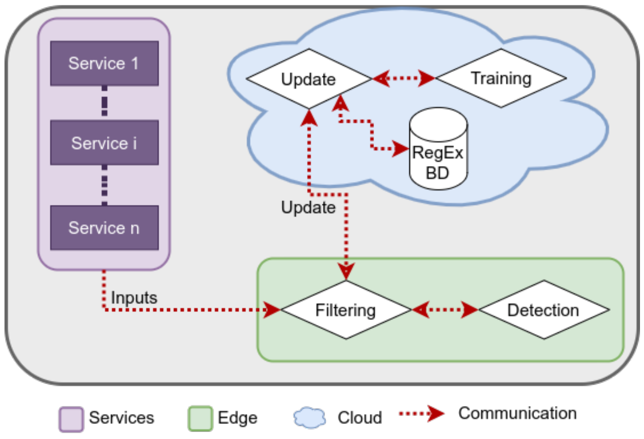

Figure 1. Overview of 2LD-SQLi

The modules are implemented as APIs, which allows access to 2LD-SQLi functionality of SQLi detection by Urban Computing services in an interoperable approach, as well as enables the communication between the edge and cloud environments through a secure channel. In the Edge environment, the *Filtering* module uses a set of RegEx to perform initial filtering of potential SQLi threats in the input data, while the *Detection* module applies an ML model previously trained in the Cloud environment. The output of the *Detection* module can be integrated with other services, such as a dashboard in a web system, notification, or other security solutions (such as IDS, Firewall, etc). On the other hand, the *Update* module stores and manages the complete set of RegEx in the RegEx DB over a cloud environment. Additionally, the *Update* module performs the update process of the ML model used in the *Detection* module at the edge environment, when it is triggered after the training process performed by the *Trainning* module. Additionally, the *Update* module controls the order of the RegEx used in the *Filtering* module, focusing on the deployment of the most suitable order in a determined situation. The flow of steps performed by 2LD-SQLi is summarized in Figure 2.

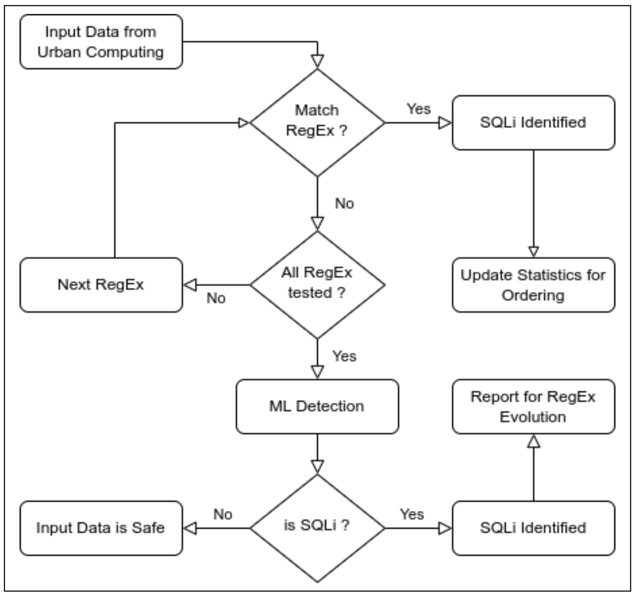

Figure 2. Steps of 2LD-SQLi

According to this organization, SQLi detection happens in the Edge environment, i.e., it occurs closer to the Urban Computing Services. The RegEx set used by 2LD-SQLi is updated through the API with the cloud environment (RegEx DB). Next, we describe in Sections 3.2 and 3.3, the Regex set applied in the *Filtering* module and the ML models used in the *Detection* module, respectively.

Finally, it is important to note that, despite 2LD-SQLi focus on SQLi detection, the designed architecture allows the proposed solution to be integrated with other security solutions, such as Intrusion Detection Systems (IDSs), Firewalls, and other security solutions. This integration enables the expansion of the applicability of 2LD-SQLi in real scenarios, where several existing threats (for example, DDoS, Side-Channel, etc). In this way, 2LD-SQLi can have its inputs monitored and applied to DDoS detection tool, which will be able to expand the security features.

3.2 RegEx in 2LD-SQLi

SQL Injection Malicious systems capitalize on weaknesses in input data, employing various tactics to deceive the system into revealing information from authorized and authenticated users. It is important to understand that SQLi has the potential to misuse username and password fields for carrying out unauthorized command-line operations. Notably, IoT devices can function as potential conduits for SQL injection (SQLi) attacks, a concern that has become more pronounced with the expansion of dedicated computing services. Each variant of SQLi introduces a distinct security breach profile, emphasizing the necessity of comprehending how SQL statements can give rise to these vulnerabilities. This endeavor seeks to conduct a prompt and effective analysis of SQLi susceptibilities by employing RegEx patterns meticulously designed to address the threat profiles that computing services could encounter.

There are several types of SQLi (tautologies, Logically Incorrect Queries, Union Queries, Inference-Based and Alternate Encoding)[**?**Das *et al.*, 2019], where we developed several RegEx to detect patterns and characteristics of each type of SQLi. The RegEx defined are:

1. Meta-characters: Search for specific SQL meta-characters to identify possible SQL injection attempts, including hexadecimal characters like single quotes, double dash, and others, that often mark the start of a comment.
   Regex: `(;%00)|(\%27)|(--[^\r\n]*)|(\')`
   - `(;%00)`: Semicolon character followed by null byte character.
   - `(\%27)`: String %27, which is the URL encoded form of the single quote character `(')`.
   - `(--[\^\r\n]*)`: Starts with two hyphens '−−' followed by any characters other than line breaks.
   - `(\')`: Quote character `(')`.
2. Alternative meta-character: Possible SQLi by looking for equal signs, single quotes, double dashes, semicolons, and the null character.
   Regex: `((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))`
   - `((%3D)|(=))`: Equal sign character or the URL encoded version of it `(%3D)`, that are used in SQL queries to denote equality in comparisons.
   - `[\^\n]*`: Any number of characters other than a newline character.
   - `((%27)|('))`: URL encoded form, or not, of single quote character.
   - `(--)`: Double hyphen string used to comment SQL queries.
   - `((\%3B)|(;))`: Semicolon character, or its URL encoded version, that separate multiple SQL statements in a single query.
3. "OR" within other strings: Search for the word 'or' in URL encoded format.
   Regex: `((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))`
   - `((\%27)|(\'))`: Simple or percent-encoded single quote.
   - `((\%6F)|o|(\%4F))`: A percent-encoded 'o', a single 'o', or a percent-encoded uppercase 'O'.
   - `((\%72)|r|(\%52))`: A percent-encoded 'r', a single 'r', or a percent-encoded uppercase 'R'.
4. Logical operators: Searches for the words 'and' or 'or' followed by spaces.
   Regex: `(\W)(and|or)\s*`
   - `(\W)`: Special character (it is not a digit, letter, or underscore).
   - `(and|or)`: 'and' or 'or'.
   - `\s*`: White spaces.
5. 'UNION' statement: Search for the word 'union' and in URL encoded format.
   Regex: `((\%27)|(\'))UNION`
   - `((\%27)|(\')`: Usual encode or percentage-encoded of a single quote.
   - `UNION`: Word 'UNION'.

6. SQL Queries: Searches for SQL keywords.
   Regex: `([\s\(\)])*(create|select|delete|update |drop|alter|insert)([\s\(\)])*`
   - `([\s\(\)])*`: Any occurrence of white space characters or parentheses.
   - `(create|select|delete|update |drop|alter|insert)`:
     One of the SQL keywords 'create', 'select', 'delete', 'update', 'drop', 'alter', or 'insert'.
7. 'exec' and 'execute': Search for keywords 'exec' or 'execute'.
   Regex: `([\s\(\)])*(execute|exec)([\s\(\)])*`
   - `([\s\(\)])*`: Any occurrence of any white space character or parentheses.
   - `(exec|execute)`: 'exec' or 'execute' keywords.
8. AND operator: Searches for AND in several formats, like plus symbol, logical operators, and encoded format.
   Regex: `(\%20and|\+and|&&|\&\&)`
   - `(\%20and)`: URL-encoded space character.
   - `(+and)`: URL-encoded space character as query parameter.
   - `(&&)` and `(\&\&)`: 'and' logical operator of programming languages.

These RegExs are designed to implement multiple pattern captures to identify a wide range of SQLi attacks. By employing several regular expressions to scrutinize the input data, the security of an application is significantly enhanced. This multi-pattern approach aids in both preventing and detecting potential SQLi attacks, as it encompasses diverse encoding methods, syntax variations, and combinations of text that attackers may use to evade security measures. Different RegEx patterns may focus on detecting SQL keywords, special characters, encoding techniques, or SQL syntax anomalies commonly associated with SQLi attacks. This comprehensive strategy ensures a more robust defense mechanism against evolving attack techniques and sophisticated intrusion attempts. Moreover, the varied nature of these regular expressions helps in overcoming limitations that may exist in individual patterns.

Additionally, the 2LD-SQLi adjusts dynamically the order of the RegEx tested in the data input, reordering it according to the matches that occurred, i.e., 2LD-SQLi brings to the front the RegEx that are hitting more SQLi threats in the filtering process (in our experiments we configured the reordering process after each 100 analysis). Dynamically adjusting the order of RegEx brings several benefits to the filtering process: Different scenarios may require different regex patterns to be applied first; The incoming data may change over time, so dynamically adjusting the regex test order can help adapt to new patterns and data structures without needing a complete reconfiguration; and, Quickly identify possible SQLi data, speeding up the filtering process and reducing the load on subsequent, more resource-intensive regex tests.

3.3 Machine Learning Model

The ML model training process involves taking in text inputs and then implementing a specific ML technique. Each ML technique employs a distinct approach to interpret the data.

Because the proposed mechanism operates independently from the existing ML technique, the following techniques are under consideration in this article: Logistic Regression (LR), is used for binary classification tasks, such as spam detection or medical diagnosis, estimating the likelihood of a binary outcome, which depends on one or more predictor variables, is the subject of investigation. Random Forest (RF) is a collective ensemble of decision trees. This approach finds application in both classification and regression tasks, as it amalgamates multiple decision trees to enhance precision and mitigate overfitting; Decision Tree (DT), is a tree-like model wherein every internal node symbolizes a feature, each branch signifies a decision rule, and each leaf node represents a particular outcome.; Convolutional Neural Network (CNN), which uses convolutional layers to automatically learn features from data, making them powerful for tasks like image classification and object detection; and, Support Vector Machine (SVM), which is designed to seek out an optimal hyperplane for effectively segregating data into distinct classes, where they are effective in high-dimensional spaces and for applications like image classification.

These ML techniques were selected based on their unique characteristics when compared to the alternatives. Different models encompass distinct algorithms, underlying assumptions, and complexities, leading to diverse behaviors and performances across datasets. Evaluating multiple models allows you to compare and contrast their performance based on various metrics such as accuracy, precision, recall, F1-score, or other relevant evaluation criteria. This comprehensive comparison not only sheds light on the strengths and weaknesses of each model but also aids in identifying the most suitable model that aligns with the specific requirements and intricacies of your problem domain. This comparison helps identify the model that has the most suitable performance for the context of SQLi detection.

4 Experiments

The repository [2] provides access to the method we've proposed. This method has been rigorously tested, as detailed in this section, to determine its capability in pinpointing SQLi threats. An authentic environment, enriched with a dataset containing SQLi specifics, was created for these tests. This environment spans various computational configurations, including both cloud-hosted virtual machines and traditional hardware systems. The deployment of 2LD-SQLi in real-world scenarios is the main focus of these tests. Insights into the performance and implications of the method are offered by this arrangement. While section 4.1 provides a comprehensive overview of the experimental design, the derived results are explored in section 4.2.

4.1 Experiment Setup

The experiments utilized test environments that included both cloud computing resources and physical machines. It

[2]https://github.com/538Michael/

is crucial to test a solution in various cloud settings to confirm its resilience, scalability, and reliability [Costa *et al.*, 2021]. We employed Huawei Cloud[3] as our cloud environment, which offers a range of Elastic Cloud Server (ECS)[4] virtual machines. Notably, the ECS in Huawei Cloud was set up with 16 GB of memory and 4 Virtual CPUs (vCPUs). Under these conditions, we considered the following cases: General Computing Plus (GCP), basic level of vCPU performance; Memory Optimized (MO), high performance of memory access; High-Performance Computing (HPC), focus on parallel computing and infrastructure services of high performance; Kunpeng (KPG), performance baseline of vCPU with Kupeng 920 processor; and, Physical Machine (PM), usual machine with CPU Intel Core i5-12400 processor, with an SSD disk and DDR4 2666MHz memory.

The experiments were based on an SQLi dataset[5] applied in several works in the literature [Rahul *et al.*, 2021; Devalla *et al.*, 2022; Hosam *et al.*, 2021; Roy *et al.*, 2022]. This dataset has 19340 entries, 7962 of which are benign entries, while 11378 of which are SQLi. Thus, it is possible to have a complete evaluation of the proposal.

A variety of machine learning techniques [6] were used, including Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), Decision Tree (DT), and Convolutional Neural Network (CNN), to benchmark 2LD-SQLi's detection ability. The time efficiency of the solution in conducting RegEx detection and updates was also assessed. For the 100 experiments, we relied on a 95% confidence interval. As such, we considered the following metrics for evaluating performance:

- Accuracy (ACC): Rate of correct classifications according to Equation 1, i.e. True Positive (TP) and True Negative (TN) cases in relation to all other cases (TP, TN, False Positive - FP and False Negative - FN).

$$Accuracy = \frac{TP + TN}{\mathrm{TP} + FN + FP + TN} \quad (1)$$

- Recall: Efficiency in correctly detecting the analyzed input, that is, the rate of TP in relation to the total number of positive cases $(TP + FN)$. Therefore, Recall is defined in Equation 2.

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + FN} \quad (2)$$

- Precision: Determines the ability to guess which of the positive values are really positive, following the definition expressed in Equation 3.

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + FP} \quad (3)$$

- F1-Score: The harmonic mean of precision and recall, i.e., the higher the precision and recall, the higher the F1-score. It is defined as in Equation 4.

$$F1 = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(FP + FN)} \quad (4)$$

[3]huaweicloud.com

[4]support.huaweicloud.com/intl/en-us/productdesc-ecs/ecs_01_0073.html

[5]kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset

[6]kaggle.com/code/chinonsocynthia/sql-inject-using-linear-models-and-cnn

- Parsing Time (milliseconds): the time required to parse an entry to determine whether it is an SQLi threat or not.

4.2 Results

Next, we discuss the experiments performed, where Figure 3 presents the processing time required to analyze input data, Figure 4 shows the detection of the stand-alone ML techniques and the RegEx step of 2LD-SQLi, and Figure 5 illustrates the performance of the whole execution of 2LD-SQLi when using several ML techniques. Finally, Tables 2 present the accuracy, F1-Score and recall evaluation metrics to compare the efficiency of 2LD-SQLi in front of existing ML techniques applied stand-alone.

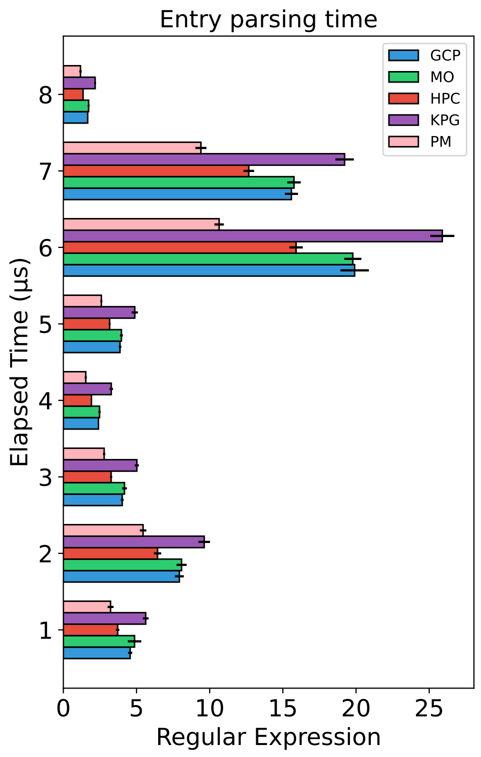

Figure 3. Time of Each RegEx to Analyze the Input Data

As shown in Figure 3, the time required for input parsing is consistent across different computational environments, encompassing both cloud virtual machines and edge physical machines. However, two key points are observed: (i) The physical machine (PM) displays a wider range of analysis times due to numerous processes and functionalities running in parallel with the 2LD-SQLi analysis process, a scenario not applicable to cloud virtual machines $(GCP, MO, HPC,$ and $KPG)$ that are dedicated to 2LD-SQLi. (ii) As outlined in Section 3.2, RegEx 1 and 5 typically take longer to process because of their more complex structures. Thus, 2LD-SQLi is concluded to have an appropriate analysis time, meeting scalability requirements by analyzing 1000 requests in around 10 milliseconds, a time significantly shorter than that of internet data communication[Gowtham and Pramod, 2022; Rizvi *et al.*, 2018].

The filtering time of 2LD-SQLi and the detection time of ML techniques are illustrated in Figure 4. The elapsed time

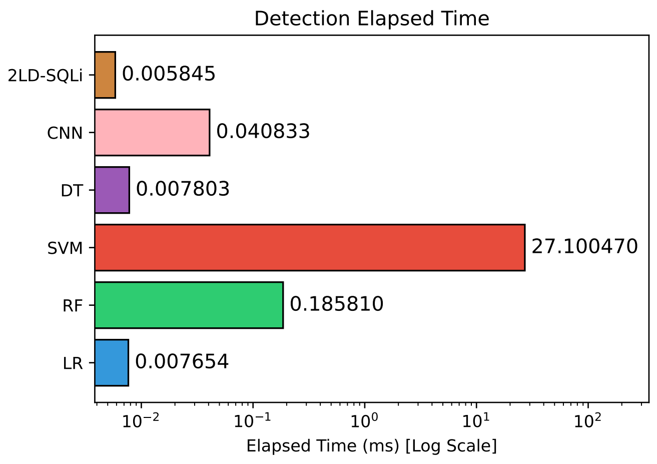

Figure 4. Filtering Time of 2LD-SQLi and Detection Time of ML Techniques

of filtering in 2LD-SQLi is about 23% faster than the fastest ML techniques (LR and DT models). This fact represents the capacity of the 2LD-SQLi to speed up the detection process since only the cases that RegEx does not capture are directed to the ML models in the Detection process of 2LD-SQLi.

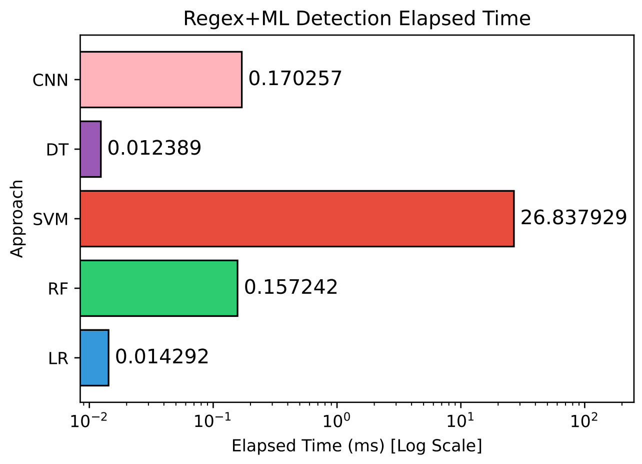

Figure 5. Detection Time of 2LD-SQLi of Both Filtering and Detection

In the experiment about processing performance, we analyzed the elapsed time of the entire process of 2LD-SQLi to detect SQLi, i.e., the elapsed time to test all the RegEx set in the *Filtering* and, when none of the RegEx result in a match, the execution of the ML model to perform the final decision. In this way, the results of Figure 5 represent the cases of both steps (RegEx and ML) when the RegEx set is not capable of identifying an SQLi (true positive) or the data input is a true negative case of SQLi. It is possible to note that in these situations, the most suitable approach is to apply the DT or LR models since they have a lower processing time. Additionally, an interesting point is the detection time of CNN model for a true negative case is smaller than the average detection time when all cases are included (as shown in Figure 4).

Additionally, Figure 6 presents the elapsed time of the detection process when the number of RegEx is incremented. Thus, it is possible to analyze the impact of higher processing (more RegEx applied) in the detection process. From this result, it is possible to note that 2LD-SQLi can keep the elapsed time with a linear behavior, i.e., the detection time proportionally follows the increase in processing load. This behavior represents the scalable features from the designed architecture integrating RegEx and ML in cloud and edge environments.

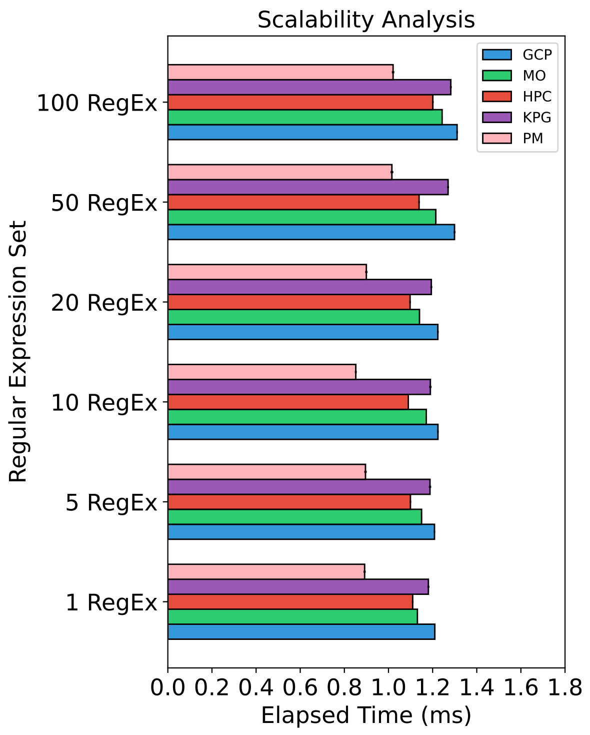

Figure 6. Scalability Analysis of 2LD-SQLi

Table 2. Detection Performance

Approach	F1_Score	Precision	Recall	ACC
LR	0.94	1.00	0.89	0.93
RF	0.94	0.99	0.89	0.93
SVM	0.81	1.00	0.68	0.82
DT	0.94	0.99	0.90	0.94
CNN	0.95	0.99	0.92	0.95
2LD-SQLi (RegEx Only)	0.98	0.99	0.97	0.98

Data shown in Table 2 illustrates the success of the evaluated solutions in identifying whether an input constitutes an SQL Injection (SQLi) or not. In the case of 2LD-SQLi, these outcomes emphasize its skill in recognizing potential threats and the value of adopting more complex solutions to save both time and computational resources.

Table 3. Detection Performance of 2LD-SQLi (RegEx+ML)

Approach	F1_Score	Precision	Recall	Accuracy
RegEx+LR	0.99	1.00	0.98	0.99
RegEx+RF	0.99	1.00	0.99	0.99
RegEx+SVM	0.98	1.00	0.97	0.98
RegEx+DT	0.99	1.00	0.99	0.99
RegEx+CNN	0.99	1.00	0.99	0.99

Finally, in Table 3, we analyze the benefits of the integration of RegEx approach and the ML models. According to

Figure 5, regarding processing, the most suitable approach is to use DT or LR models in the Detection module of 2LD-SQLi. Similarly, when the data presented in Table 3 is analyzed, it is possible to conclude that the most suitable option is to use the DT model in 2LD-SQLi, which reaches the highest values in all evaluated metrics.

4.3 Final discussion

Based on the results of the experiments performed, it is possible to note that the efficiency of 2LD-SQLi, regarding both response time and detection efficiency, overcomes the existing approach to applying stand-alone ML to detect SQLi. Therefore, deploying 2LD-SQLi as a cybersecurity tool for the first line of defense in Edge environments can be crucial to support urban computing services since it has a low response time and high accuracy in the detection process.

In real-world applications, especially Urban Computing services that handle massive amounts of data, scalability becomes paramount. The scalable SQL injection detection performed by 2LD-SQLi can efficiently process the inputs, ensuring that no potential threats go unnoticed even in high-traffic situations. Additionally, the high accuracy of 2LD-SQLi ensures that security teams can trust the alerts generated by it, leading to efficient utilization of resources for investigating and mitigating real threats. This is crucial for compliance standards but also instills trust among users and stakeholders that their data is protected against malicious exploitation.

Another important issue is the arising of new Cyber threats, including SQLi, that are constantly evolving, where the adjustable set of RegEx for filtering with high accuracy can be evolved naturally to new attack vectors, patterns, and evasion techniques, ensuring continued protection against emerging threats without compromising performance or reliability.

An existing limitation of 2LD-SQLi is the evolution of RegEx set based on the report of a situation where a SQLi is detected by ML but it was not filtered by the RegEx set. Currently, the analysis of the report needs to be performed manually by the security team to generate a new RegEx to encompass this case. Thus, a possible evolution of 2LD-SQLi is the automation of RegEx generation based on the report using, for example, Generative Artificial Intelligence (GAN) [Ye *et al.*, 2020; Chen *et al.*, 2020].

5 Conclusion and Future Work

Urban Computing services have become a reality in recent years due to the expansion of communication and information technologies throughout cities and metropolitan centers. These services generate a massive volume of data stored in SQL DBs. This scenario increased the possibility of threats, mainly SQLi. To address this reality, this article introduces 2LD-SQLi, an SQLi threat detection solution based on RegEx that aims to act as an initial filtering service for protection against SQLi threats to meet response time and scalability requirements. Subsequently, inputs considered as threats are analyzed by a machine learning model.

As future work, we intend to develop a new RegEx to be added to the set of the filtering process. Additionally, we will perform data mining in the SQLi dataset, applying clustering algorithms to understand better the profile of the inputs, as well as the capacity of the solution to detect each group. As the final evolution, we intend to perform an automatic update and evolution of RegEx used, applying new techniques such as GAN.

Declarations

Acknowledgements

The authors would like to thank the National Council for Scientific and Technological Development (CNPq) of Brazil (N^o 303877/2021-9) and the Latin America and Caribbean Network Information Center (LACNIC) through the FRIDA Grant for the financial support.

Authors' Contributions

Author Michael Silva contributed to the conception of this study, analysis, and writing. He is the main contributor and writer of this manuscript. Author Silvio Ribeiro and Author Rafael Gomes participated in the validation of the study, review and final editing. Finally, Author Vanessa Carvalho and Francisco José read and approved the final manuscript.

References

Chen, Q., Wang, X., Ye, X., Durrett, G., and Dillig, I. (2020). Multi-modal synthesis of regular expressions. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 487–502, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3385412.3385988.

Costa, W. L., Portela, A. L., and Gomes, R. L. (2021). Features-aware ddos detection in heterogeneous smart environments based on fog and cloud computing. *International Journal of Communication Networks and Information Security*, 13(3):491–498. Available at:`https://www.proquest.com/openview/47831cccd3eca021e332e78d816f9227/1?pq-origsite=gscholar&cbl=52057`.

Crespo-Martínez, I. S., Campazas-Vega, A., Guerrero-Higueras, Á. M., Riego-DelCastillo, V., Álvarez-Aparicio, C., and Fernández-Llamas, C. (2023). Sql injection attack detection in network flow data. *Computers & Security*, 127:103093. DOI: 10.1016/j.cose.2023.103093.

da Silva, G., Oliveira, D., Gomes, R. L., Bittencourt, L. F., and Madeira, E. R. M. (2020). Reliable network slices based on elastic network resource demand. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. DOI: 10.1109/NOMS47738.2020.9110316.

Das, D., Sharma, U., and Bhattacharyya, D. K. (2019). Defeating sql injection attack in authentication security: an experimental study. *International Journal of Information Security*, 18(1):1–22. DOI: 10.1007/s10207-017-0393-x.

Devalla, V., Srinivasa Raghavan, S., Maste, S., Kotian, J. D., and Annapurna, D. D. (2022). murli: A tool for detection of malicious urls and injection attacks. *Procedia Computer Science*, 215:662–676. 4th International Conference on Innovative Data Communication Technology and Application. DOI: 10.1016/j.procs.2022.12.068.

Fadolalkarim, D., Bertino, E., and Sallam, A. (2020). An anomaly detection system for the protection of relational database systems against data leakage by application programs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 265–276. DOI: 10.1109/ICDE48307.2020.00030.

Geldenhuys, M. K., Will, J., Pfister, B. J. J., Haug, M., Scharmann, A., and Thamsen, L. (2021). Dependable iot data stream processing for monitoring and control of urban infrastructures. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 244–250. DOI: 10.1109/IC2E52221.2021.00041.

Gomes, R. L., Bittencourt, L. F., and Madeira, E. R. M. (2020). Reliability-aware network slicing in elastic demand scenarios. *IEEE Communications Magazine*, 58(10):29–34. DOI: 10.1109/MCOM.001.2000753.

Gowtham, M. and Pramod, H. B. (2022). Semantic query-featured ensemble learning model for sql-injection attack detection in iot-ecosystems. *IEEE Transactions on Reliability*, 71(2):1057–1074. DOI: 10.1109/TR.2021.3124331.

Hosam, E., Hosny, H., Ashraf, W., and Kaseb, A. S. (2021). Sql injection detection using machine learning techniques. In *2021 8th International Conference on Soft Computing Machine Intelligence (ISCMI)*, pages 15–20. DOI: 10.1109/ISCMI53840.2021.9654820.

Li, Q., Li, W., Wang, J., and Cheng, M. (2019). A sql injection detection method based on adaptive deep forest. *IEEE Access*, 7:145385–145394. DOI: 10.1109/ACCESS.2019.2944951.

Lv, Z., Hu, B., and Lv, H. (2020). Infrastructure monitoring and operation for smart cities based on iot system. *IEEE Transactions on Industrial Informatics*, 16(3):1957–1962. DOI: 10.1109/TII.2019.2913535.

Moreira, D. A. B., Marques, H. P., Costa, W. L., Celestino, J., Gomes, R. L., and Nogueira, M. (2021). Anomaly detection in smart environments using ai over fog and cloud computing. In *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*, pages 1–2. DOI: 10.1109/CCNC49032.2021.9369449.

Musznicki, B., Piechowiak, M., and Zwierzykowski, P. (2022). Modeling real-life urban sensor networks based on open data. *Sensors*, 22(23). DOI: 10.3390/s22239264.

Parashar, D., Sanagavarapu, L. M., and Reddy, Y. R. (2021). Sql injection vulnerability identification from text. In *14th Innovations in Software Engineering Conference (Formerly Known as India Software Engineering Conference)*, ISEC 2021, New York, NY, USA. Association for Computing Machinery. DOI: .

Portela, A. L., Menezes, R. A., Costa, W. L., Silveira, M. M., Bittecnourt, L. F., and Gomes, R. L. (2023). Detection of iot devices and network anomalies based on anonymized network traffic. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6. DOI: 10.1109/NOMS56928.2023.10154276.

Portela, A. L. C., Ribeiro, S. E. S. B., Menezes, R. A., de Araujo, T., and Gomes, R. L. (2024). T-for: An adaptable forecasting model for throughput performance. *IEEE Transactions on Network and Service Management*, pages 1–1. DOI: 10.1109/TNSM.2024.3349701.

Rahul, S., Vajrala, C., and Thangaraju, B. (2021). A novel method of honeypot inclusive waf to protect from sql injection and xss. In *2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, volume 1, pages 135–140. DOI: 10.1109/CENTCON52345.2021.9688059.

Rizvi, S., Kurtz, A., Pfeffer, J., and Rizvi, M. (2018). Securing the internet of things (iot): A security taxonomy for iot. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 163–168. DOI: 10.1109/TrustCom/BigDataSE.2018.00034.

Rodrigues, D. O., Santos, F. A., Filho, G. P. R., Akabane, A. T., Cabral, R., Immich, R., Junior, W. L., Cunha, F. D., Guidoni, D. L., Silva, T. H., Rosário, D., Cerqueira, E., Loureiro, A. A. F., and Villas, L. A. (2019). Computação urbana da teoria à prática: Fundamentos, aplicações e desafios.

Roy, P., Kumar, R., and Rani, P. (2022). Sql injection attack detection by machine learning classifier. In *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, pages 394–400. DOI: 10.1109/ICAAIC53929.2022.9792964.

Silveira, M. M., Portela, A. L., Menezes, R. A., Souza, M. S., Silva, D. S., Mesquita, M. C., and Gomes, R. L. (2023). Data protection based on searchable encryption and anonymization techniques. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–5. DOI: 10.1109/NOMS56928.2023.10154280.

Souza, M., Ribeiro, S., and Gomes, R. (2023). Detecção de ameaças de injeção de sql em serviços de computação urbana. In *Anais do VII Workshop de Computação Urbana*, pages 145–158, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/courb.2023.801.

Tang, P., Qiu, W., Huang, Z., Lian, H., and Liu, G. (2020). Detection of sql injection based on artificial neural network. *Knowledge-Based Systems*, 190:105528. DOI: 10.1016/j.knosys.2020.105528.

Xie, X., Ren, C., Fu, Y., Xu, J., and Guo, J. (2019). Sql injection detection for web applications based on elastic-pooling cnn. *IEEE Access*, 7:151475–151481. DOI: 10.1109/ACCESS.2019.2947527.

Ye, X., Chen, Q., Wang, X., Dillig, I., and Durrett, G. (2020). Sketch-Driven Regular Expression Generation from Natural Language and Examples. *Transactions of the Association for Computational Linguistics*, 8:679–694. DOI: 10.1162/tacl$_{a0}$0339.