Computers & Security 127 (2023) 103093

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

SQL injection attack detection in network flow data

Ignacio Samuel Crespo-Martínez[a,*], Adrián Campazas-Vega[b], Ángel Manuel Guerrero-Higueras[b], Virginia Riego-DelCastillo[b], Claudia Álvarez-Aparicio[b], Camino Fernández-Llamas[b]

[a] Supercomputación Castilla y León (SCAYLE), Campus de Vegazana s/n, León 24071, Spain
[b] Robotics Group, University of León, Campus de Vegazana s/n, León 24071, Spain

ARTICLE INFO

Article history:
Received 1 August 2022
Revised 4 December 2022
Accepted 3 January 2023
Available online 5 January 2023

Keywords:
Ensamble learning
Machine learning
Netflow
Network security
SQLIA detection

ABSTRACT

SQL injections rank in the OWASP Top 3. The literature shows that analyzing network datagrams allows for detecting or preventing such attacks. Unfortunately, such detection usually implies studying all packets flowing in a computer network. Therefore, routers in charge of routing significant traffic loads usually cannot apply the solutions proposed in the literature. This work demonstrates that detecting SQL injection attacks on flow data from lightweight protocols is possible. For this purpose, we gathered two datasets collecting flow data from several SQL injection attacks on the most popular database engines. After evaluating several machine learning-based algorithms, we get a detection rate of over 97% with a false alarm rate of less than 0.07% with a Logistic Regression-based model.

1. Introduction

Cyber-attacks are a growing concern for companies, organizations, and users. The number of cyber-attacks and the diversity of techniques used have increased exponentially in recent years. Web applications are among the most exploited attack vectors. Such applications offer a wide variety of functionality, allowing users to consult data, perform banking operations or even make purchases over the Internet. Significant security problems of web infrastructures are injections, especially SQL injections. An SQL injection attack (SQLIA) allows an intruder to interact with a web application's database, stealing information or even modifying or deleting legitimate data stored in the application (Clarke, 2009). Open Web Application Security Project (OWASP) is a worldwide non-profit project seeking to improve software security. This community publishes the "OWASP Top 10", a standard awareness document for developers and web application security. It represents a broad consensus about web applications' most critical security risks (Foundation, 2022a). OWASP Top 10 ranks injections as the third most serious web application security risk in the Top 10 published in 2021. Similarly, MITRE (2022b) publishes the CWE Top 25 Most Dangerous Software Weaknesses (MITRE, 2022a). In this ranking, SQLIAs also occupy third place.

Detecting SQLIAs are addressable issues if all network-layer datagrams were to be analyzed. However, some networks handle such a large amount of information that it is impossible to explore the contents of every network packet. Therefore, SQLIA detection is an unresolved problem in this type of network. Such networks usually use lightweight protocols based on network streams such as NetFlow, sFlow or IPFIX.

This work presents several contributions. First, two labelled datasets have been gathered and published under a CC Attribution 4.0 International license. They collect network flow data from SQLIAs on the three most widely-used relational database engines. Besides, several supervised learning-based models have been fitted with the above datasets. The results show that it is possible to detect SQLIAs using NetFlow Version 5-based flow data. Finally, for each model, a comprehensive set of metrics is provided to confirm the validity of the trained models.

The remainder of the paper is organized as follows: Section 2 describes some key findings in the literature; Section 3 describes the materials and tools employed in this work, as well as the methodology used to collect the datasets, to set every model's hyperparameters, and to evaluate the experiments;

* Corresponding author.
E-mail addresses: icrem@unileon.es (I.S. Crespo-Martínez), acamv@unileon.es (A. Campazas-Vega), am.guerrero@unileon.es (Á.M. Guerrero-Higueras), vriec@unileon.es (V. Riego-DelCastillo), calvaa@unileon.es (C. Álvarez-Aparicio), camino.fernandez@unileon.es (C. Fernández-Llamas).

https://doi.org/10.1016/j.cose.2023.103093

Section 4 shows the results obtained in the experiments and Section 5 discusses them. Finally, the conclusions are presented in Section 6.

2. Related work

It has been shown in the literature that it is possible to detect SQLIAs using machine-learning approaches. Ordered from oldest to newest, in Uwagbole et al. (2017), the authors collected a dataset containing the extraction of known attack patterns, including SQL tokens and symbols presented at injection points. The authors fitted a Support Vector Machine (SVM)-based model to the above dataset, resulting in an accuracy of 98.6% and an $\mathcal{F}_1$*_score* of 98.5%. In Ross et al. (2018), the authors gathered three datasets containing SQLIAs collecting network traffic from two points: (1) at the web application host and (2) at an appliance node located between the web app host and the associated MySQL database server. The third dataset merges data from the previous datasets. The authors trained jRip-, J48-, Random Forest (RF)-, SVM-based models, and an Artificial Neural Network (ANN) to experiment. The best results were obtained on the merged dataset with a 98.05% accuracy using the RF model and a 97.61% accuracy using the ANN. The study in Zhang (2019) presents a machine-learning classifier designed to identify SQL injection vulnerabilities in PHP code from features extracted from text strings and subsequently normalized using the TF-IDF bag-of-words algorithm. The authors obtained the best results using an SVM with an accuracy score of 95.4% and a Detection Rate ($\mathcal{DR}$) of 98.6; a Convolutional Neural Network (CNN) also yields a good performance with an accuracy score of 95.3% and a $\mathcal{DR}$ of 95.4%. In Hasan et al. (2019), the authors focused on testing many algorithms to identify which ones offered the best results in SQLIA detection. The results showed that out of the 23 models; the five best-performing algorithms were the Boosted Trees and Bagged Trees ensemble, the Linear Discriminant (LD), and two SVM-based models. The ensemble methods showed the best results, with an accuracy of 93.8%. In work carried out in Tripathy et al. (2020), the authors focused only on the payload of the packets to try to detect SQLIA. To do so, they trained several supervised learning-based models. The model that showed the best results was RF with an accuracy score of 99.8%, followed by the Boosted Tree Classifier, Adaptive Boosting Classifier (AdaBoost), Decision Tree (DT), and SGD Classifier models, all with an accuracy score of over 98.6%. The research conducted in Farooq (2021) attempts to detect SQLIA by splitting queries into their corresponding tokens and then applying algorithms to the tokenized dataset. The authors used only ensemble machine learning algorithms, namely Gradient Boosting Machine (GBM), AdaBoost, Extended Gradient Boosting Machine (XGBM), and Light Gradient Boosting Machine (LGBM). The experiment was a success. All the models tested by the authors obtained an accuracy score and a $\mathcal{DR}$ higher than 99%. In Roy et al. (2022), the authors used the Kaggle SQL Injection dataset with multiple machine learning methods to identify and detect SQLIA. The best performing model was Naive Bayes (NB) with an accuracy score of 98.3%, followed by Logistic Regression (LR) with an accuracy score of 92.7%. In Deriba et al. (2022), the authors proposed a comprehensive framework to determine the efficacy of the proposed techniques for dealing with a range of problems depending on the type of attack, using a hybrid approach (statistical and dynamic) and machine learning. The results showed that the hybrid approach obtained an accuracy of 99.2%. The ANN and SVM models also performed well, with an accuracy score of 98.5% and 96.8%, respectively.

All of the above work demonstrates that it is possible to detect SQLIA using machine-learning. Datasets containing network-layer packets to train this model type is one of the most widely used approaches. Network-layer datagrams store all the information exchanged in network-layer communication, not only the headers but also the payload. Therefore, networks with a high traffic load can only analyze some packets in-depth. Such networks often use lightweight flow-based protocols to get valuable data for monitoring network activity. A flow is a set of packets passing through an observation point in the network during a specific time interval. All packets within the same flow have common features such as IP addresses and ports, both source and destination (Claise et al., 2013). Flow data do not gather the payload of the packets. It considerably reduces the computational load required to process flows versus complete network packets. As a result, the use of flows is widespread in networks that need to reduce their routers' computational load. The most common flow-based technology is NetFlow, specifically NetFlow V5. This technology is implemented in routers from well-known companies such as Cisco Systems (2022), Juniper Networks (2022), and Enterasys Switches Networks (2022).

It has been shown in the literature that it is possible to detect specific network attacks such as Denial of Service (DoS) attacks or port scans using machine learning models trained with network flows. The research carried out in Kemp et al. (2018) tried to detect application-layer Distributed Denial of Service (DDos) attacks (specifically Slow Read attacks) using NetFlow data. They used eight classification algorithms to build Slow Read attack-detection models. The authors concluded that six of their predictive models perform well at detecting such attacks; the Random Forest model is the one that offered the best results with an Area Under the Curve (AUC) of 96,8%. Working with port scanning attacks in Campazas-Vega et al. (2020), we have presented DOROTHEA, a tool that generates tagged flow data suitable for fitting classification models using supervised learning algorithms. The authors gathered two flow datasets with port scanning attacks and benign traffic. The first dataset (used to fit the models) gathered regular port scans. The second dataset (used to test the models) gathered slow port scans. The results showed that the models K-Nearest Neighbors (KNN) and Logistic Regression (LR) obtained an accuracy higher than 94%, certifying that it is possible to detect port scanning attacks using detection models with flow data. According to the above results, we can conclude that the features extracted from a network flow are discriminant enough to distinguish between benign traffic and some network attacks.

The literature shows that SQLIA detection using full network-layer datagrams is an addressable problem. However, detecting SQLIA is very difficult when using network flow data since flows do not retain the payload of the packets. Although some works detect network attacks using flow-based protocols, detecting SQLIA using network flow data is an unsolved problem.

In work, Sarhan et al. (2020) the authors generated flow-based datasets from well-known packet-based public datasets and then compared the performance of the Extra Trees ensemble classifier in the packet-based dataset versus the flow-based dataset generated. nProbe tool by nTop (2022) was utilized by the authors to convert the Pcaps in NetFlow V9 format. They selected 12 features to be extracted from the Pcap files. The datasets were evaluated in two ways, 1) Binary format (attack or non-attack) and 2) multiple classes depending on the type of attack recorded. Focusing on the NF-CSE-CIC-IDS2018 dataset, which is the only one that contains a class that stored only SQLIA, the results obtained using a binary classification and the proposed model through this dataset were a detection rate of 94.71% and an $\mathcal{F}_1$*_score* of 83%. Using a multiclass classification and the proposed model through the flow-based dataset, the results obtained for SQLIA were a detection rate of 25.00% and an $\mathcal{F}_1$*_score* of 22%. These results show that the only research that analyzes SQL injections using network flows data does not have a detection rate higher than 25%.Therefore, SQLIA detection using network flows is a problem that has not yet been solved.

3. Materials and methods

This section presents the materials and experiments performed and the methods used to evaluate them. First, we will review what SQLIA is, the mechanisms to carry it out, and the types of SQLIA. Next, we will detail what NetFlow technology is. Then, we propose guidelines for the collection of suitable flow datasets and present the treatment of the data used by the datasets. Finally, we propose the evaluation method.

3.1. SQL injection attack

An SQLI is a security vulnerability in which an attacker interferes with an application's queries on its database. In general, it allows an attacker to see specific data that he should not be able to retrieve. For example, it might include data belonging to other users, logins, table structures, or any other data the application can access. In many cases, an attacker can also modify or delete such data, causing persistent changes to the content or behaviour of the application. In some situations, it is also possible that an attacker can escalate an SQLIA to compromise the underlying server or other back-end infrastructure or perform denial of service attacks (Junjin, 2009). The 3 most-used database management systems (DBMS) are MySQL (Oracle, 2022), PostgreSQL (Group, 2022), and Microsoft SQL Server (Microsoft, 2022).

3.1.1. SQLI Mechanism

SQLI vulnerabilities can be found in any application parameter used in a database query through which the SQLIA can be initiated. There are different ways through which an attacker can compromise databases. This method is known as an injection mechanism (Chandrashekhar et al., 2012). There are four types of injection mechanisms:

1. *Injection through cookies.* Cookies contain information generated by web applications and stored in the client. When the client returns to an application, cookies are used to restore the client's state information. A malicious client could alter the content of cookies. For example, if a web application uses the contents of cookies to create SQL queries, an attacker could easily send an attack by embedding the injection in the vulnerable cookie.
2. *Injection through the user input.* It is usually the most common mechanism. The user's input is not controlled and is included directly in a SQL statement.
3. *Injection through server variables.* Server variables are a collection of variables containing network headers and environment variables. Web applications use these server variables in various ways, such as recording usage statistics and identifying browsing trends. If these variables are recorded in a database without sanitization, this could create an SQLI vulnerability.
4. *Second-order or stored injections.* These attacks are the most complex and challenging to detect. They consist of an attack in two phases. In the first phase, ”part” of the necessary content is inserted to carry out the attack, which will be executed in the second phase. E.g. when registering on a web server, you use the following username “`admin'--`”. Once logged in, the attacker will modify the password of the newly created user. The SQL statement will look something like this:

   ```
   UPDATE users SET password='newpwd' WHERE
   userName='admin'--' AND password='oldpwd'
   ```

   Since “`--`” in SQL refers to starting comments, everything after it will be ignored, and the attacker will modify the administrator's password.

3.1.2. SQLIA Types

We will look at the main types of SQLIA according to Jemal et al. (2020):

- *Error attack.* It is the most common attack, and the easiest to exploit since it is the application that indicates the database errors when performing the different queries.
- *Union attack.* This attack consists of the portal returning a result and then adding the result of another query to the actual result in such a way as to display, together with the portal data, the sensitive data of the portal that should not be available.
- *Blind attack.* This is the most complicated and advanced attack and is the last option when none of the previous attacks work. In this case, the attacker has to be very creative and ask questions to the database questions using booleans, i.e. true or false, to get the information he needs to know.
  - *Based on conditional.* This technique is based on trying to get a different response from the web application based on a particular condition. If this condition is TRUE, the response is loaded correctly. If this condition is FALSE, the web application shows an error message or will not return anything.
  - *Time based.* This technique consists of sending an SQL command to the server with code to force a delay of n seconds in the execution of queries. If the query is successful, it will return the results after n seconds; if not, it will not display anything.

3.2. NetFlow

NetFlow (Claise et al., 2004) is a lightweight protocol developed by Cisco Systems to collect flow data. NetFlow has become popular in networks that handle large amounts of traffic. Manufacturers such as Juniper and Enterasys Switches support this technology. NetFlow was introduced as a new feature of Cisco routers to collect IP traffic. This functionality gives administrators a global view of what is happening on the network they manage. NetFlow has many versions: NetFlow V1, V5 and V9. The most widely used version of NetFlow is Version 5. For this version, the features collected are listed in Table 1.

NetFlow generates unidirectional flows. So in network communication, it builds two flows: one flow gathers packets with a source-destination address, and a second one gathers reply packets with a destination-source address.

A NetFlow stream expires after a time of inactivity or when it has been active for more than a specific amount of time. Both time periods can be configured. Although expiring an active flow may be contradictory; these flows are terminated so that the flow analyzer can obtain information on long-lived flows, thus preventing “infinite” flows from not being analyzed.

3.3. DOROTHEA

DOROTHEA (Campazas-Vega et al., 2020) – a tool developed by the authors – is a Docker-based framework for NetFlow data collection. It allows one to build interconnected virtual networks to generate and collect flow data using the NetFlow protocol. DOROTHEA sents network traffic packets to a NetFlow generator with a sensor *ipt_netflow* installed (Aabc/IPT-netflow, 2022). It consists of a module for the Linux kernel, which processes the packets and converts them to NetFlow flows data.

It is customizable and scalable. Furthermore, it allows for deploying nodes that generate synthetic network traffic, both benign and malicious.

Benign traffic generation nodes simulate network traffic generated by real users, performing tasks such as searching in web browsers, sending emails, or establishing Secure Shell (SSH) connections. Such tasks run as Python (Foundation, 2022b) scripts.

Table 1
NetFlow V5 Features.

Feature	Description
sysuptime	Current time in milliseconds since the export device started
unix_secs	Current count of seconds since 0000 UTC 1970
unix_nsecs	Residual nanoseconds since 0000 UTC 1970
engine_type	Flow switching motor type
engine_id	Slot number switching engine flow
exaddr	Flow exporter IP
srcaddr	Source IP address
dstaddr	Destination IP address
nexthop	IP address of the next hop router
input	SNMP index of the input interface
output	SNMP index of the exit interface
dpkts	Number of packets contained in the flow
doctets	Total number of bytes of layer 3 in the packets of the flow
first	Sysuptime at start of flow
last	Sysuptime when the last packet in the flow was received
srcport	TCP / UDP source port number
dstport	TCP / UDP destination port number
tcp_flags	TCP flags
prot	IP type of protocol (e.g., TCP = 6; UDP = 17)
tos	IP type of service (ToS)
src_as	Autonomous system number of the source, either source or pair
dst_as	Autonomous system number of the destination, either source or pair
src_mask	Source address prefix mask bits
dst_mask	Destination address prefix mask bits

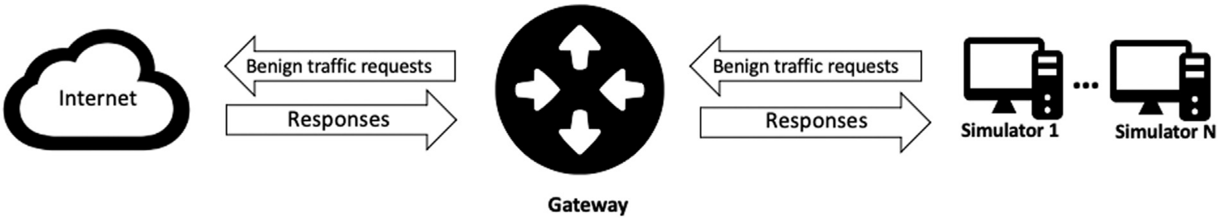


Fig. 1. Benign traffic generation scheme.

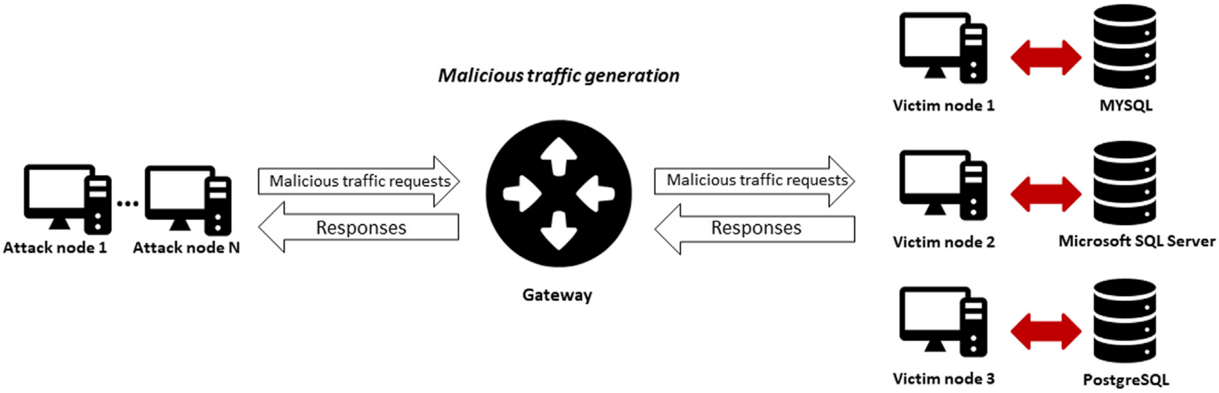


Fig. 2. Malicious traffic generation scheme.

Users may customize them or even incorporate their own. The network traffic is managed by a gateway that performs two main tasks. On the one hand, it routes packets to the Internet. On the other hand, it sends it to a NetFlow data generation node (this process is carried out similarly to packets received from the Internet). Figure 1 shows the benign traffic generation process.

Regarding the malicious traffic generation process, DOROTHEA uses a similar approach as in the benign traffic generation, as shown in Fig. 2. In this case, the environment isolates, so all traffic is labelled empirically as malicious. The attacks run as Python scripts, just as in the benign traffic generation process. DOROTHEA also allows the user to customize such scripts or add new ones. The gateway works as explained above by routing packets and gathering flow data.

3.4. Data gathering

Two Netflow V5 datasets have been collected with DOROTHEA. Besides, It finishes a flow after it is inactive for 15 s or after it is active for 1800 s (30 min). The first dataset ($\mathcal{D}_1$) was collected to train the detection models, and the second one ($\mathcal{D}_2$) to test them. The datasets gather flow data from different attacks to ensure their generalization. These datasets are published, and available online (Campazas-Vega and Crespo-Martínez, 2022).

The datasets contain both benign and malicious traffic. Both datasets are balanced. The percentage of malicious and benign traffic is around 50% to prevent the classifiers from always predicting the majority class. As the malicious traffic corresponds to SQLIA, all malicious traffic is sent through ports 443 and 80. Approxi-

Table 2
SQLMAP execution parameters.

Parameters	Description
`--banner, --current-user, --current-db, --hostname, --is-dba, --users, --passwords, --privileges, --roles, --dbs, --tables, --columns, --schema, --count, --dump, --comments, --schema`	Enumerate users, password hashes, privileges, roles, databases, tables and columns
`--level=5`	Increase the probability of a false positive identification
`--risk=3`	Increase the probability of extracting data
`--random-agent`	Select the User-Agent randomly
`--batch`	Never ask for user input, use the default behavior
`--answers=’’follow=Y’’`	Predefined answers to yes

Table 3
Dataset volumetry and distribution.

Dataset	Aim	#samples	Benign-malicious traffic rate	Class	% of src. or dst. port 80 or 443
$\mathcal{D}_1$	Training	400,003	50%	Benign	56.02%
				Malicious	100%
$\mathcal{D}_2$	Test	57,229	50%	Benign	59.78%
				Malicious	100%

mately 60% of the benign traffic of both datasets corresponds to traffic with source or destination on ports 80 or 443. A high percentage of benign traffic has been collected through ports 80 and 443 to prevent the models from being distorted by port numbers. The volume of data is shown in Table 3.

The benign flow data in both datasets were generated using Python scripts. These simulate network traffic generated by real users, performing tasks such as searching web browsers, sending emails, and establishing SSH connections.

The malicious traffic collected in $\mathcal{D}_1$ (SQLI attacks), was performed using SQLMAP. SQLMAP is a penetration tool used to automate the process of detecting and exploiting SQL injection vulnerabilities (Ojagbule et al., 2018). SQL The attacks were executed on 16 nodes and by launching SQLMAP with the parameters shown in the Table 2. Every node executed SQLIA on 200 victim nodes. The victim nodes had deployed a web form vulnerable to Union-type injection attacks, which was connected to the MYSQL or SQLServer database engines (50% of the victim nodes deployed MySQL and the other 50% deployed SQLServer). The web service was accessible from ports 443 and 80, which are the ports typically used to deploy web services. The IP address space was 182.168.1.1/24 for the benign and malicious traffic-generating nodes. For victim nodes, the address space was 126.52.30.0/24.

The malicious traffic in the test sets was collected under different conditions. For $\mathcal{D}_1$, SQLIA was performed using Union attacks on the MySQL and SQLServer databases. However, for $\mathcal{D}_2$, Blind-SQL SQLIAs were performed against the web form connected to a PostgreSQL database. The IP address spaces of the networks were also different from those of $\mathcal{D}_1$. In $\mathcal{D}_2$, the IP address space was 152.148.48.1/24 for benign and malicious traffic generating nodes and 140.30.20.1/24 for victim nodes.

DOROTHEA requires separating the benign-malicious traffic generation process to empirically label the flow as benign or malicious so that the traffic is not collected simultaneously.

3.5. Data processing

Data processing has been carried out to improve the performance of the models and eliminate the bias they may have due to the nature of the data generated. The data processing techniques used in this study consist of three steps:

Feature cleaning

First, the IP addresses are converted to a numeric value, and the datasets are checked for empty columns or rows to avoid errors in the generation of the models.

Dimensionality reduction

This technique is used to reduce the complexity of the models. For each feature, the models' complexity increases exponentially, which decreases the detection capacity.

Netflow V5 has 24 features; the feature variance has been computed to decrease their number. Variance is a measure of dispersion that is used to represent the variability of a set of data concerning its arithmetic mean (Scheffe, 1999). After applying the variance of the features of the benign flows with those of the malicious flows in $\mathcal{D}_1$ and $\mathcal{D}_2$, the features ‘exaddr’, ‘engine_type’, ‘engine_id’, ‘src_mask’, ‘dst_mask’, ‘src_ac’, and ‘dst_as’ have been removed. The variance of these features was 0. Besides the above features, the following features have been removed: ‘unix secs’, ‘unix_nsecs’, ‘sysuptime’, ‘first’, and ‘last’. These characteristics are time-related. These features have been removed so that the models are not biased depending on when the data was collected. Finally, the “nexthop” feature has been removed. This feature negatively influences the detection of malicious traffic in wide area networks, and needs to be removed in the training phase of machine learning models (Campazas-Vega et al., 2021).

Data normalization

It is necessary to normalize the value of the features to a specific range to analyze the data accurately. This technique is done to avoid errors related to the scale of the data. For example, the size of an IP address is around ten digits, but the number of packets in a flow is usually no more than two digits. Therefore, if the data is not normalized, the models can generate a bias by interpreting the IP address as having more weight.

In this work, the min-max-based linear data normalization technique has been used. The minimum-maximum normalization is shown in Eq. (1) where X‘s are the values to be normalized and $Min(A)$ and $Max(A)$ are the minimum and maximum values of features (A) before normalization. After applying this normalization, all the data that make up $\mathcal{D}_1$ and $\mathcal{D}_2$ are on a scale between ‘0’ and ‘1’.

$$\text{Min-max} = \frac{X - Min(A)}{Max(A) - Min(A)} \tag{1}$$

3.6. Classification models fitting

MoEv was used to fit the detection models. It is just a wrapper for the Scikit-learn API. MoEv allows automatically building classification models from labelled datasets using the Scikit-learn library by editing a YAML-based configuration file (Pedregosa et al., 2011).

MoEv features include data preprocessing-cleaning, normalization, dimensionality reduction, and hyperparameter tuning through the GridSearchCV class. Besides, it generates a report that provides relevant information such as accuracy, false alarm rate, Matthews correlation coefficient, Cohen's kappa coefficient, detection rate, recall, and $\mathcal{F}_1$*_score*. MoEv has been used successfully in different research areas, such as detecting jamming attacks in real-time location systems (Guerrero-Higueras et al., 2018), and predicting the students' academic success (Guerrero-Higueras et al., 2020). In addition, in Campazas-Vega et al. (2020) MoEv has been successfully used to build malicious traffic detection models on flow data.

We want to predict a category – benign (0) or malicious (1) – so classification algorithms are more suitable than regression or clustering algorithms. However, since data matters more than algorithms for complex problems (Banko and Brill, 2001; Halevy et al., 2009), we aim to evaluate both classification and regression algorithms to select the most accurate for this problem. We used MoEv to fit, tune the hyperparameters, and finally test our detection models.

Specifically, the following algorithms were used: KNN (Mitchell and Schaefer, 2001), LR (Wright, 1995), Linear Support Vector Classification (LSVC) (Cortes and Vapnik, 1995), Perceptron with stochastic gradient descent (SGD) (Bottou, 1991), and RF (Breiman, 2001).

In addition to the above models, the classification process was performed through an ensemble-based classification technique, which uses a combined action of the classifiers listed above for malicious traffic detection. The main idea of this ensemble process is to activate the strengths of each algorithm to achieve a robust classifier. Majority voting (VC) is the type of ensemble classifier used. The ensemble chooses the class that receives the highest number of votes, regardless of whether the sum of those votes exceeds 50% (Krishnaveni and Prabakaran, 2021).

The proposed ensemble classification model and the individual models have been trained with $\mathcal{D}_1$ and tested with $\mathcal{D}_2$.

3.7. Evaluation

The confusion matrix allows for computing well-known Key Performance Indicators (KPIs) to identify the most accurate classification algorithm. First, the models' performance was measured using their accuracy score on the test sets computed as shown in Eq. (2). T_P is the number of malicious flows correctly identified as malicious. T_N points to the number of benign flows correctly identified as benign traffic. F_P is the number of benign samples incorrectly classified as malicious. Finally, F_N points out the number of malicious samples wrongly classified as benign traffic.

$$\text{Accuracy} = \frac{T_P + T_N}{T_P + F_P + T_N + F_N} \tag{2}$$

Furthermore, the following KPIs have been considered: False Alarm Rate (FAR) and Matthews correlation coefficient (ϕ). Moreover, since binary classifiers tend to predict the majority class, we also compute Detection Rate ($\mathcal{DR}$), Recall ($\mathcal{R}$), and $\mathcal{F}_1$*_score* for both classes – benign (0) and malicious (1) flow data.

The FAR is calculated as the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events (regardless of classification). It computes as shown in Eq. (3).

$$\text{FAR} = \frac{F_P}{T_N + F_P} \tag{3}$$

ϕ is often used to measure the quality of binary classifiers. It computes as shown in Eq. (4).

$$\phi = \frac{T_P \times T_N - F_P \times F_N}{\sqrt{(T_P + F_P)(T_P + F_N)(T_N + F_P)(T_N + F_N)}} \tag{4}$$

$\mathcal{DR}$ measures the accuracy of the positive predictions. It computes as shown in Eq. (5).

$$\mathcal{DR} = \frac{T_P}{T_P + F_P} \tag{5}$$

$\mathcal{R}$, also called sensitivity or true positive rate, is the ratio of positive instances correctly detected by the classifier. It computes as shown in Eq. (6).

$$\mathcal{R} = \frac{T_P}{T_P + F_N} \tag{6}$$

It is often convenient to combine detection rate and recall into a single metric called the $\mathcal{F}_1$*_score* ($\mathcal{F}_1$), in particular, if a simple way to compare two classifiers is needed. $\mathcal{F}_1$ is the harmonic mean of $\mathcal{DR}$ and $\mathcal{R}$. Whereas the regular mean treats all values equally, the harmonic mean gives much more weight to low values. It computes as shown in Eq. (7).

$$\mathcal{F}_1 = 2\frac{\mathcal{DR} \times \mathcal{R}}{\mathcal{DR} + \mathcal{R}} \tag{7}$$

In addition, Cohen's kappa coefficient (κ) has been computed (Cohen, 1960). Cohen's kappa computes a score expressing the agreement level between two raters in a classification problem. It is computed as shown in Eq. (8) where P_o is the empirical probability of agreement on the label assigned to any sample, and P_e is the expected agreement when both annotators assign labels randomly. P_e is estimated using a per-annotator empirical prior to the class labels (Artstein and Poesio, 2008).

$$\kappa = \frac{P_o - P_e}{1 - P_e} \tag{8}$$

4. Results

First, we want to point out that a Jupyter Notebook that allows for replicating the evaluation is available online in a Binder-ready repository.[1]

To detect SQLIA in flow data NetFlow V5 format using machine learning models, dataset $\mathcal{D}_1$ was used to train the models, and dataset $\mathcal{D}_2$ was used to test the models to ensure that they had generalization capability. After tuning, the following hyperparameters were selected for each model:

- LR. We used a regularized version of linear regression, specifically Ridge regression. This model does not apply penalties; the algorithm used in the optimization problem is 'sag'.
- LSVC. We used a linear kernel function. The regularization parameter (C) is set to 1.0. As a loss function, we used Squared Hinge. The norm used in the penalization is 'l2'.
- Perceptron+SGD. Stochastic gradient descent has been applied to the perceptron algorithm. No penalties are applied. The constant that multiplies the regularization term is set to 0.0001.
- RF. We trained with 80 trees in the forest. The minimum number of samples required to split an internal node is 0.1, and the minimum number of samples needed at a leaf node is 2.
- KNN. We used 1 neighbour for the neighbour queries. The distance metric used for the tree is 'minkowski'. The algorithm used to compute the nearest neighbours is 'ball_tree'

Figure 3 shows the confusion matrices created from the predicted data of the trained models using the dataset $\mathcal{D}_2$. The values of the main diagonal correspond to the values correctly estimated by the model(T_P and T_N). The other diagonal represents the cases in which the model failed in its prediction(F_P and F_N). The numbers add up to the 57,229 $\mathcal{D}_2$ flows. Models mostly fail in prediction giving false positives. Presenting respectively the confusion matrix for

[1] https://github.com/uleroboticsgroup/MoEv/tree/SQLInjection.

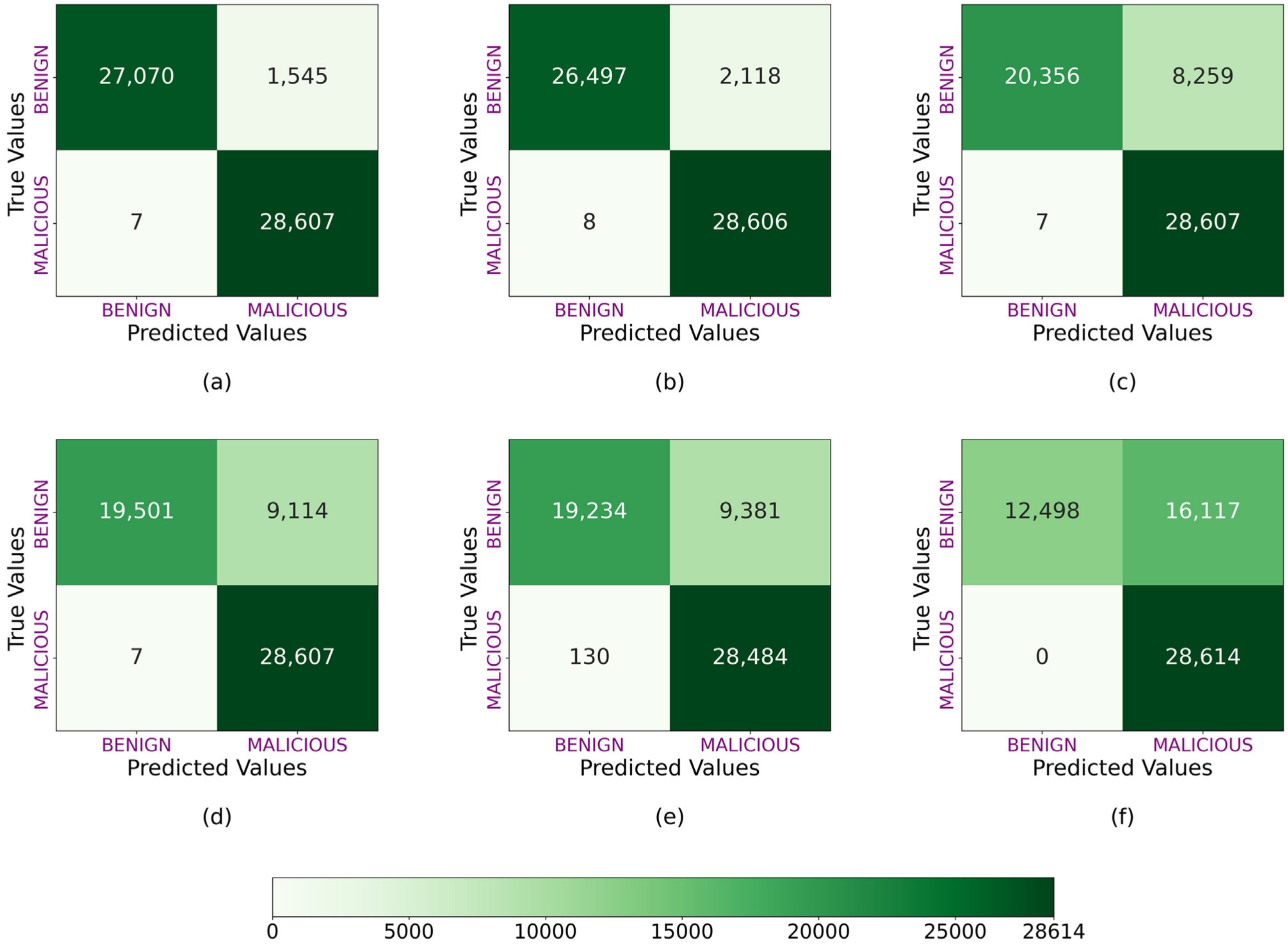


Fig. 3. Confusion matrix for LR-, Perceptron+SGD-, VC-, RF-, LSVC- and KNN- detection models (from left to right) using the dataset $\mathcal{D}_2$.

Table 4
Accuracy, Detection Rate, Recall and $\mathcal{F}_1$_*score* obtained in Campazas-Vega et al. (2020) for malicious-traffic detection models on flow data.

Algorithm	Class	$\mathcal{DR}$	$\mathcal{R}$	$\mathcal{F}_1$
LR	**Benign (0)**	**0.999**	**0.946**	**0.972**
	Malicious (1)	**0.949**	**0.999**	**0.974**
	Average	**0.974**	**0.973**	**0.973**
Perceptron+SGD	Benign (0)	0.999	0.926	0.961
	Malicious (1)	0.931	0.999	0.964
	Average	0.965	0.963	0.963
VC	Benign (0)	0.999	0.711	0.831
	Malicious (1)	0.776	0.999	0.874
	Average	0.888	0.856	0.852
RF	Benign (0)	0.999	0.681	0.810
	Malicious (1)	0.758	0.999	0.862
	Average	0.879	0.840	0.836
LSVC	Benign (0)	0.993	0.672	0.802
	Malicious (1)	0.752	0.995	0.857
	Average	0.873	0.834	0.829
KNN	Benign (0)	1.000	0.437	0.608
	Malicious (1)	0.639	1.000	0.780
	Average	0.819	0.718	0.694

(a) LR, (b) Perceptron+SGD, (c) VC, (d) RF, (e) LSVC and (f) KNN. Several KPIs (Accuracy, ϕ and FAR) were computed from the confusion matrices in Fig. 3. Besides, κ was computed as shown in Eq. (8). Figure 4 shows accuracy, ϕ, κ and FAR for the LR, Perceptron+SGD, VC, RF-based, LSVC, and KNN malicious-traffic detection models. Finally, Table 4 shows the $\mathcal{DR}$, $\mathcal{R}$, and $\mathcal{F}_1$ for the same models. In this case, we computed the KPIs for both classes – benign and malicious flow data – to identify classifiers that tend to predict the majority class.

5. Discussion

Focusing on the accuracy scores shown in Fig. 4, the best models are LR and Perceptron+SGD, with an accuracy score higher than 96% in both cases. The model that offers the third best accuracy is the ensemble model built by hard voting with an accuracy score higher than 87%. The LSVC and RF models also obtain acceptable results with an accuracy score above 83% in both models. However, the model that offers the worst results is the KNN, with an accuracy score lower than 72%. Regarding ϕ, κ and FAR of the studied models follow the same trend. LR and Perceptron+SGD obtain the best ϕ, κ with values higher than 92% in both models. The ensemble model and the LSVC and RF models obtain an acceptable ϕ and κ with values close to 70%. Finally, the worst model is the KNN model with ϕ lower than 52% and a κ lower than 44%. A model provides better results the lower the FAR is. LR and Perceptron+SGD demonstrate the lowest FAR rate with values below 1%. The VC, LSVC, and RF models obtain a FAR close to 21%. Finally, the KNN model has a very high FAR of 36%.

The remaining KPIs ($\mathcal{DR}$, $\mathcal{R}$, and $\mathcal{F}_1$) show the same tendency. Table 4 shows $\mathcal{DR}$, $\mathcal{R}$, and $\mathcal{F}_1$ higher than 94% in the LR model for both malicious (1) and benign traffic (0) and values higher than 92% in the Perceptron+SGD model for both types of traffic. The cumulative value of these indicators is higher than 97% in the LR model and 96% in the Perceptron+SGD model. These data confirm

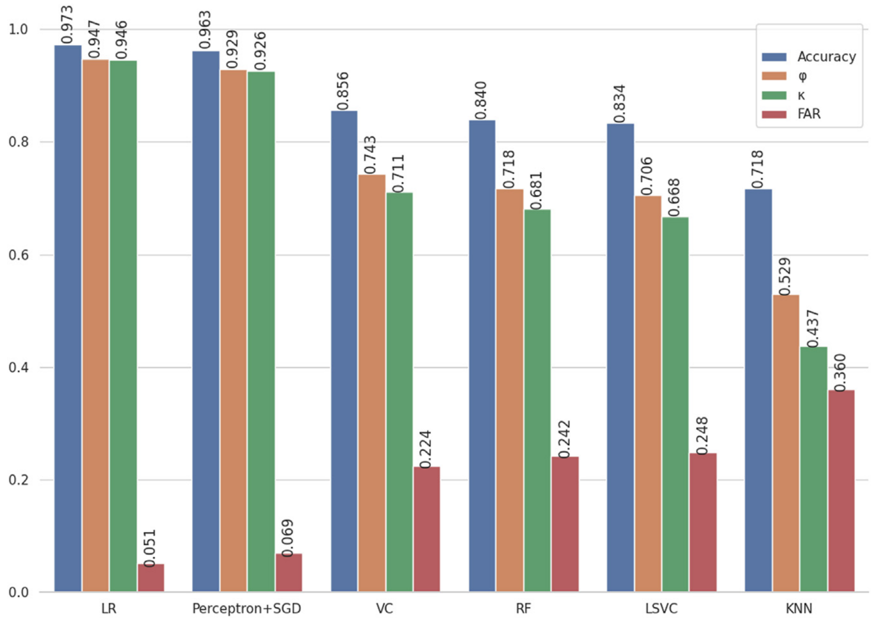


Fig. 4. Models metrics.

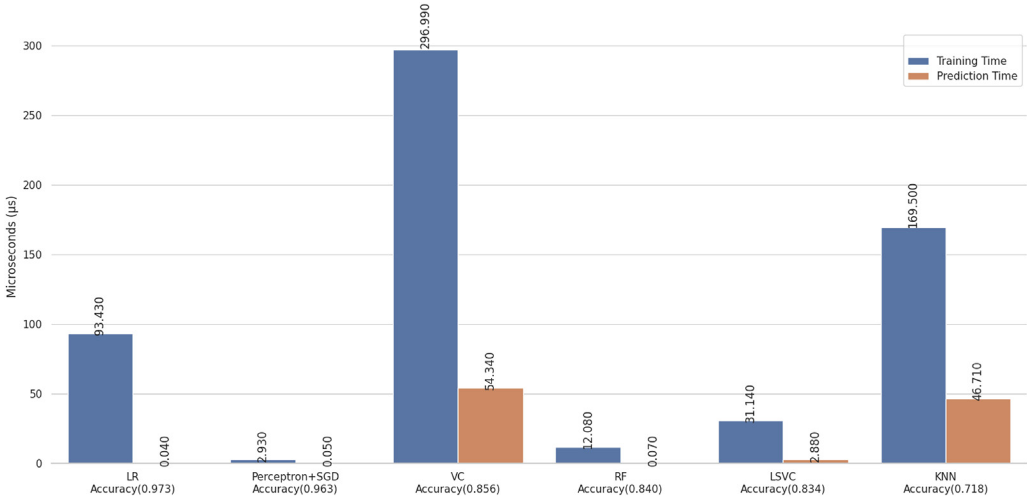


Fig. 5. Model execution times.

that these models do not discriminate between classes and show a high detection capability for both malicious and benign traffic.

The VC, RF and LSVC models obtain a cumulative $\mathcal{DR}$ of both classes higher than 87%. These models offer differences between classes, given that the $\mathcal{DR}$ for benign traffic (0) is close to 99% in all three models. However, the $\mathcal{DR}$ corresponding to malicious traffic drops to values close to 75%. $\mathcal{R}$ shows a similar trend in these models, with malicious traffic (1) being the dominant value. Finally, the $\mathcal{F}_1$ shows cumulative values above 82% in the three models, with the assembled model being the best $\mathcal{F}_1$ above 85%. Finally, the worst performing model is the KNN with a cumulative $\mathcal{DR}$ below 81% and $\mathcal{R}$ and $\mathcal{F}_1$ below 72%.

From the results obtained, it can be concluded that the LR and Perceptron+SGD models offer the best results for the overall proposed indicators. Therefore, it can be asserted that these models can detect SQL injections in flow data, specifically in NetFlow V5 flows, which are currently the most used ones. Besides, it can also be stated that the ensemble-generated model (VC) and the RF and LSVC models show promising results in SQLIA detection, with the VC model being the best of the 3.

Regarding execution times, as shown in the Fig. 5, in the training phase the model that takes the longest to train is the VC, since it encompasses the execution of all models, followed by the KNN and LR. The fastest model in the training phase is the Perceptron+SGD. In the prediction phase, the LR and Perceptron+SGD models stand out with a prediction time of less than 0.05 per sample in both cases, being also the two best models in this aspect.

As mentioned above in Section 1 there are no researches that directly address the SQLIA detection in flow data. Only the work done in Sarhan et al. (2020) obtains a $\mathcal{DR}$ to 25% and it's $\mathcal{F}_1$ to 22% in detecting SQLInjection with the Extra Tree model on the NF-CSE-CIC-IDS2018. In order to compare our work with the work presented in Sarhan et al. (2020), we started from the same dataset used by the authors (CSE-CIC-IDS2018) and obtained the packets belonging to SQLIA. Subsequently, using the softflowd (Miller, 2022) tool, NetFlow V5 flows have been generated from the network packets in Sarhan et al. (2020) the authors used Nprobe, this tool has been discarded in this work as it is a paid tool). Subsequently, the data have been treated as shown in Section 3, performing the same operations that were carried out on datasets

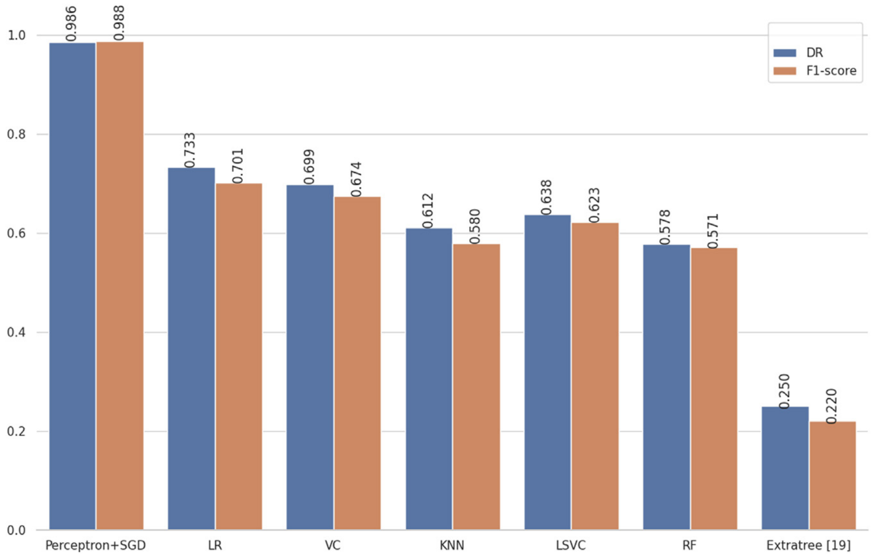


Fig. 6. Comparison of models with (Sarhan et al., 2020).

$\mathcal{D}_1$ and $\mathcal{D}_2$. Finally, the models trained with dataset $\mathcal{D}_1$ with the parameterization reflected in Section 4 have been validated against the new dataset generated. As can be seen in Fig. 6, the Perceptron-SGD model demonstrates the best results with a $\mathcal{DR}$ and $\mathcal{F}_1$ higher than 98%, a result much higher than the 25% of DR and the 22% of $\mathcal{F}_1$ obtained by the authors in Sarhan et al. (2020). In addition, the models LR, KNN, and the voting ensemble model obtain a $\mathcal{DR}$ and a $\mathcal{F}_1$ of almost 70%, demonstrating a good generalization capacity.

6. Conclusions

Machine-learning models using complete network packets are a common and valid solution for detecting SQLIA. In conventional-sized local area networks, packet payload analysis allows the detection of such traffic. However, packet analysis is not feasible in wide area networks, where the amount of network traffic is significant. These networks typically use lightweight flow-based protocols like NetFlow to collect traffic statistics. SQLIA detection is currently an unresolved problem in this type of network. This paper has generated and published two datasets based on NetFlow V5 flows. The generated datasets contain different SQLI executed on the currently most-used database engines.

Machine-learning-based models train and test with the above datasets – specifically, LR-, Perceptron+SGD-, LSVC-, RF-, KNN detection models, and an ensemble classifier based on majority voting (VC).

The LR and Perceptron+SGD models have shown promising results, with an accuracy and a $\mathcal{DR}$ higher than 96% in both cases. Besides, these two models have demonstrated a FAR of less than 1%, which confirms the SQLIA detection capability of these models. In addition to the above models, the model generated as an assembly of the rest of the models (VC) has also demonstrated a high SQLIA detection capability with an accuracy score of 85.6% and DR of 89%

The results have shown that detecting SQLIA attacks in networks is possible using NetFlow as a lightweight, flow-based protocol.

This work has been a starting point for detecting SQLIA in flow data. The proposed models can be deployed in production, detecting SQL injections on network flow data and generating alerts to improve the security of users, companies or administrations. In future works, we intend to increase the network range, including even IPv6.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Ignacio Samuel Crespo-Martínez: Conceptualization, Software, Validation, Investigation, Data curation, Writing – original draft, Visualization. **Adrián Campazas-Vega:** Software, Investigation, Data curation, Writing – review & editing. **Ángel Manuel Guerrero-Higueras:** Conceptualization, Methodology, Validation, Investigation, Writing – review & editing, Visualization, Project administration. **Virginia Riego-DelCastillo:** Software, Investigation, Writing – review & editing. **Claudia Álvarez-Aparicio:** Software, Investigation, Writing – review & editing. **Camino Fernández-Llamas:** Conceptualization, Methodology, Resources, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Data availability

SQL Injection Attack Detection in Network Flow Data.

Acknowledgments

This research has been partially supported under the grant Detección de nuevas amenazas y patrones desconocidos en la Red Regional de Ciencia y Tecnología", addenda 4 and 8 to the framework agreement between Instituto Nacional de Ciberseguridad de España (INCIBE) and Universidad de León 2019–2022 funded by INCIBE; and under the grant PID2021-126592OB-C21 funded by MCIN/AEI/ 10.13039/501100011033 and by the European Union".

References

Aabc/IPT-netflow, 2022. Ipt-netflow: netflow iptables module for linux kernel. https://github.com/aabc/ipt-netflow (accessed July 28, 2022).

Artstein, R., Poesio, M., 2008. Inter-coder agreement for computational linguistics. Comput. Linguist. 34 (4), 555–596.

Banko, M., Brill, E., 2001. Scaling to very very large corpora for natural language disambiguation. In: Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, pp. 26–33.

Bottou, L., 1991. Stochastic gradient learning in neural networks. Proc. Neuro-Nimes 91 (8), 12.

Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32.

Campazas-Vega, A., Crespo-Martínez, I.S., Guerrero-Higueras, Á.M., Álvarez-Aparicio, C., Matellán, V., 2021. Analysis of netflow features' importance in malicious network traffic detection. In: Computational Intelligence in Security for Information Systems Conference. Springer, pp. 52–61.

Campazas-Vega, A., Crespo-Martínez, I.S., Guerrero-Higueras, Á.M., Fernández-Llamas, C., 2020. Flow-data gathering using netflow sensors for fitting malicious–traffic detection models. Sensors 20 (24), 7294.

Campazas-Vega, A., Crespo-Martínez, I. S., 2022. SQL Injection Attack Netflow (D1-D2). 10.5281/zenodo.6907251. Online; accessed July 26, 2022.

Chandrashekhar, R., Mardithaya, M., Thilagam, S., Saha, D., 2012. SQL injection attack mechanisms and prevention techniques. In: Thilagam, P.S., Pais, A.R., Chandrasekaran, K., Balakrishnan, N. (Eds.), Advanced Computing, Networking and Security. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 524–533.

Cisco Systems, I., 2022. Cisco. https://www.cisco.com/ (accessed July 26, 2022).

Claise, B., Sadasivan, G., Valluri, V., Djernaes, M., 2004. Cisco systems netflow services export version 9. RFC 3954. Internet Engineering Task Force.

Claise, B., Trammell, B., Aitken, P., 2013. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information, RFC 7011 (Internet Standard), Internet Engineering Task Force, 2070–1721.

Clarke, J., 2009. SQL Injection Attacks and Defense. Elsevier.

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educ. Psychol. Meas. 20 (1), 37–46.

Cortes, C., Vapnik, V., 1995. Support vector machine. Mach. Learn. 20 (3), 273–297.

Deriba, F. G., SALAU, A. O., Mohammed, S. H., Kassa, T. M., Demilie, W. B., 2022. Development of a compressive framework using machine learning approaches for SQL injection attacks. Przeglad Elektrotechniczny.

Farooq, U., 2021. Ensemble machine learning approaches for detection of SQL injection attack. Tehnički glasnik 15 (1), 112–120.

Foundation, O., 2022. Owasp top ten. https://owasp.org/www-project-top-ten/ (accessed July 20, 2022).

Foundation, P. S., 2022. Python. https://www.python.org/(accessed July 26, 2022).

Group, P. G. D., 2022. Postgresql. https://www.postgresql.org/ (accessed July 26, 2022).

Guerrero-Higueras, Á.M., DeCastro-García, N., Matellán, V., 2018. Detection of cyber-attacks to indoor real time localization systems for autonomous robots. Rob. Auton. Syst. 99, 75–83.

Guerrero-Higueras, Á.M., Fernández Llamas, C., Sánchez González, L., Gutierrez Fernández, A., Esteban Costales, G., González, M.Á.C., 2020. Academic success assessment through version control systems. Appl. Sci. 10 (4), 1492.

Halevy, A., Norvig, P., Pereira, F., 2009. The unreasonable effectiveness of data. IEEE Intell. Syst. 24 (2), 8–12.

Hasan, M., Balbahaith, Z., Tarique, M., 2019. Detection of SQL injection attacks: a machine learning approach. In: 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA). IEEE, pp. 1–6.

Jemal, I., Cheikhrouhou, O., Hamam, H., Mahfoudhi, A., 2020. SQL injection attack detection and prevention techniques using machine learning. Int. J. Appl. Eng.Res. 569–580.

Juniper Networks, I., 2022. Juniper. https://www.juniper.net/ (accessed July 26, 2022).

Junjin, M., 2009. An approach for SQL injection vulnerability detection. In: 2009 Sixth International Conference on Information Technology: New Generations, pp. 1411–1414. doi:10.1109/ITNG.2009.34.

Kemp, C., Calvert, C., Khoshgoftaar, T., 2018. Utilizing netflow data to detect slow read attacks. In: 2018 IEEE International Conference on Information Reuse and Integration (IRI). IEEE, pp. 108–116.

Krishnaveni, S., Prabakaran, S., 2021. Ensemble approach for network threat detection and classification on cloud computing. Concurrency Comput. Pract.Exp. 33 (3), e5272.

Microsoft, 2022. Sqlserver. https://www.microsoft.com/en-us/sql-server/ (accessed July 26, 2022).

Miller, D., 2022. softflowd. https://github.com/irino/softflowd (accessed September 12, 2022).

Mitchell, H., Schaefer, P., 2001. A "soft" k-nearest neighbor voting scheme. Int. J. Intell. Syst. 16 (4), 459–468.

mitre, 2022. 2022 CWE top 25 most dangerous software weaknesses. https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html (accessed September 13, 2022).

mitre, 2022. mitre. https://www.mitre.org/ (accessed September 13, 2022).

Networks, E., 2022. Extreme networks. https://www.extremenetworks.com/ (accessed July 26, 2022).

Ntop, 2022. Nprobe. https://www.ntop.org/products/netflow/nprobe/ (accessed September 9, 2022).

Ojagbule, O., Wimmer, H., Haddad, R.J., 2018. Vulnerability analysis of content management systems to SQL injection using SQLMAP. In: SoutheastCon 2018. IEEE, pp. 1–7.

Oracle, 2022). Mysql. https://www.mysql.com/ (accessed July 26, 2022).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.

Ross, K., Moh, M., Moh, T.-S., Yao, J., 2018. Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection. In: Proceedings of the ACMSE 2018 Conference, pp. 1–8.

Roy, P., Kumar, R., Rani, P., 2022. SQL injection attack detection by machine learning classifier. In: 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC). IEEE, pp. 394–400.

Sarhan, M., Layeghy, S., Moustafa, N., Portmann, M., 2020. NetFlow datasets for machine learning-based network intrusion detection systems. In: Big Data Technologies and Applications. Springer, pp. 117–135.

Scheffe, H., 1999. The Analysis of Variance, Vol. 72. John Wiley & Sons.

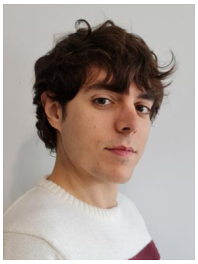
Tripathy, D., Gohil, R., Halabi, T., 2020. Detecting SQL injection attacks in cloud SaaS using machine learning. In: 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS). IEEE, pp. 145–150.

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

Wright, R.E., 1995. Logistic regression. Reading and Understanding Multivariate Statistics.

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

Ignacio S. Crespo Martinez got his degree in computer science (2017) and his master of science (2019) in Cybersecurity research at University of León (León, Spain). He is currently pursuing the PhD in computer science at the University of León. From 2018 to 2020 he has worked as a cybersecurity auditor. In 2020, he became part of a cybersecurity research project for the University of Leon.



Adrián Campazas Vega got his degree in computer science (2015) and his master of science (2019) in Cybersecurity research at University of León (León, Spain). He is currently pursuing the PhD in computer science at the University of León. From 2015 to 2019 he has worked in different companies as an application developer and cybersecurity auditor. In 2020, he became part of a cybersecurity research project for the University of Leon. In addition, during the academic year 2019–2022 he has been associate lecturer in the area of computer architecture at the University of Leon.

Ángel Manuel Guerrero Higueras got his degree (2007) and his master of science (2010) in computer science at Rey Juan Carlos University (Madrid, Spain). Besides, he got his PhD at the University of León in 2017. He worked as an IT engineer at several companies in the private sector from 2000 to 2010 and from 2014 to 2016. In academia, he worked as a research assistant in the Atmospheric Physics Group (2011–2013) and in the Research Institute of Applied Science to Cyber- Security (2016–2018), both depending on the University of León. He currently stands as an Assistant Professor at the University of León. His main research interests include cybersecurity, robotics and AI.

Virginia Riego-DelCastillo got the Computer Science degree from the University of León (Spain) in 2018 and the master in 2020. She is currently pursuing the PhD in computer science at University of León. Since 2015, she has been collaborating at the University of León (Spain) in the Department of Mechanical, Computer Science and Aerospace Engineering. Her research interests are centered on high performance computing and digital image processing and analysis applied to industrial processes and Precision Livestock Farming.

Claudia Álvarez Aparicio got her Computer Science Degree in 2017 and her Cybersecurity research Master's in 2019. Now she is Assistant Professor and she is currently working on her PhD thesis in computer science at the University of León. From 2017 to 2018, and from 2019 to July 2020, she has been a Research Associate with the Research Institute of Applied Science to Cyber-Security and the Robotics Group at the Universidad de León. From July 2020 to September 2021, she had a fellowship provided by the Regional Government of Castilla y León (Spain) to work on her Ph.D. Her research interests include computer security and social robotics.

Camino Fernández-Llamas Dr. Camino Fernández got her Computer Science Engineering degree (1994), a MSc on Knowledge and Software Engineering (1995) and her PhD on Computer Science (2000) by the Polytechnic University of Madrid, Spain. She started her teaching and researching career at University Carlos III of Madrid in 1995, and since 2008 she works for the University of León (Spain). Part of the Robotics Research Group, her main research interests are artificial intelligence and more specifically, human-machine and human-robot interaction, haptic simulation and cybersecurity. She has co-authored around 200 works in journals, conferences and workshops, and has been part or led more than 30 projects and contracts on these subjects.