Journal of Information Security and Applications 85 (2024) 103871

Contents lists available at ScienceDirect

Journal of Information Security and Applications

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

SQL injection attack: Detection, prioritization & prevention

Alan Paul, Vishal Sharma, Oluwafemi Olukoya *

School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, United Kingdom

ARTICLE INFO

Keywords:
SQL injection
Web application security
HTTP requests
NetFlow
Network security
Machine learning
Threat intelligence
Vulnerability prioritization
Database attack
Deep learning
Risk modelling

ABSTRACT

Web applications have become central in the digital landscape, providing users instant access to information and allowing businesses to expand their reach. Injection attacks, such as SQL injection (SQLi), are prominent attacks on web applications, given that most web applications integrate a database system. While there have been solutions proposed in the literature for SQLi attack detection using learning-based frameworks, the problem is often formulated as a binary, single-attack vector problem without considering the prioritization and prevention component of the attack. In this work, we propose a holistic solution, SQLR34P3R, that formulates the SQLi attack as a multi-class, multi-attack vector, prioritization, and prevention problem. For attack detection and classification, we gathered 457,233 samples of benign and malicious network traffic, as well as 70,023 samples that had SQLi and benign payloads. After evaluating several machine-learning-based algorithms, the hybrid CNN-LSTM models achieve an average F1-Score of 97% in web and network traffic filtering. Furthermore, by using CVEs of SQLi vulnerabilities, SQLR34P3R incorporates a novel risk analysis approach which reduces additional effort while maintaining reasonable coverage to assist businesses in allocating resources effectively by focusing on patching vulnerabilities with high exploitability. We also present an in-the-wild evaluation of the proposed solution by integrating SQLR34P3R into the pipeline of known vulnerable web applications such as Damn Vulnerable Web Application (DVWA) and Vulnado and via network traffic captured using Wireshark from SQLi DNS exfiltration conducted with SQLMap for real-time detection. Finally, we provide a comparative analysis with state-of-the-art SQLi attack detection and risk ratings solutions.

1. Introduction

Web applications are frequently targeted by threat actors due to their rapid production and wide attack surface. A recent security audit from Astra Security [1] states that a web application faces a cyber-attack every 39 s. The surge in web applications and smartphone usage, coupled with the increasing technical proficiency of end users, has led to a spike in web traffic. Businesses migrating their systems from offline to online systems has provided threat actors with an increased attack surface to exploit. The Open Worldwide Application Security Project (OWASP) lists SQL Injection (SQLi) among the top three web application security risks in both the 2017 and 2021 versions [2]. An SQLi attack is a web attack that is used to target data stored in database management systems (DBMS) by injecting malicious input, which is directly concatenated with original SQL queries issued by the client application to subvert application functionality and perform unauthorized operations. An example usage of SQLi attack to bypass authentication is demonstrated in Fig. 1, which displays the threat actor injecting the SQLi payload "admin'OR 1=1 – -" into a "user" field of a login page using any arbitrary password. The malicious input, "admin'OR 1=1 – -", which always evaluates to true, is then directly concatenated to the prefixed SQL statement with the "– -" commenting out the remainder of the SQL bypassing password verification and successfully authenticating the threat actor to the web application.

The widespread accessibility of specialized SQLi exploitation frameworks, along with the increasing attack surface of web applications, makes it challenging to offer complete protection against SQLi attacks. The lack of secure coding practice among developers introduces numerous SQLi-related vulnerabilities, compromising the confidentiality, integrity, and availability (CIA) of the data stored within DBMS systems. To compromise confidentiality, SQLi can be used to access sensitive information stored in the database servers by web applications such as personal identifiable information (PII), health and financial information. Threat actors can use standard SQL commands such as "UPDATE" and "ALTER" to make changes to the database records to compromise the integrity of the data, while the use of commands such as "DELETE" and "DROP" to delete individual records or the entire table respectively compromises the availability of the data.

* Corresponding author.
E-mail addresses: apaul06@qub.ac.uk (A. Paul), v.sharma@qub.ac.uk (V. Sharma), o.olukoya@qub.ac.uk (O. Olukoya).

https://doi.org/10.1016/j.jisa.2024.103871

Available online 31 August 2024

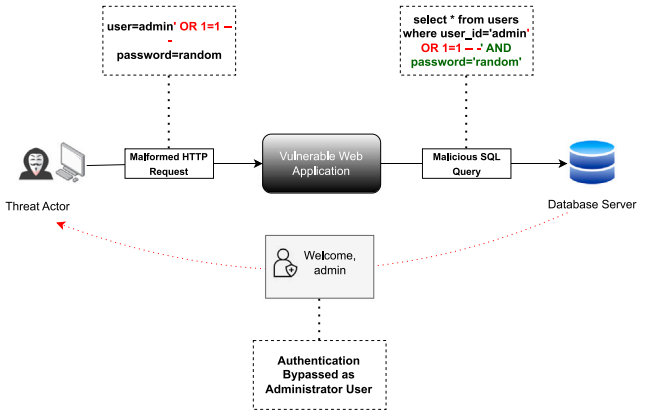


Fig. 1. An example of SQL injection.

Between 2021 and 2024, different types of SQLi attacks have been used in many high-profile data breaches in several platforms such as WooCommerce [3], BillQuick [4], Django [5], MOVEit [6], Fortra FileCatalyst Workflow [7] and Ivanti Endpoint Manager (EPM) [8]. These attacks have resulted in data theft, malware deployment, data exfiltration, CIA compromise, authentication bypass, remote code execution and, in some cases, reputational damage, and regulatory fines. The majority of the SQLi attacks occur through the URL and request body parameters of GET and POST requests, respectively. However, threat actors have started to exploit alternative web application parameters such as HTTP header fields (including Cookies, X-Forwarded-For, User-agent and Referer) [9]. The out-of-band category of SQLi attacks uses non-standard protocols like DNS to retrieve data through alternative channels [10]. The emergence of various SQLi detection and prevention solutions has prompted threat actors to adapt their strategies, leading to the execution of complex SQLi attacks performed by chaining different attack types and advanced evasion techniques such as Denial of service, DNS hijacking and DNS data exfiltration [11–13].

According to MITRE [14], SQL Injection is ranked #3 in the 2023 CWE Top 25 most dangerous software weaknesses. Additionally, MITRE analysed 15 vulnerabilities that appeared on every list during the previous five publications (2019–2023). SQL Injection is one of the vulnerabilities that keep coming up in the CWE Top 25 [15]. This indicates that SQL injection (SQLi) vulnerabilities persist in commercial and open-source software, as confirmed by CISA, FBI and OWASP foundation [16,17]. Although automated tools exist for identifying and taking advantage of SQL injection vulnerabilities in web applications, such as SQLMap, their ability to detect certain SQL Injection techniques may be limited [18]. Vulnerability scanners and built-in sanitization frameworks are excellent for automating bug hunting, fuzzing, hacking and penetration testing, they are also prone to inaccuracy, inconsistency, and false positives in reported output [19,20]. Using a learning-based framework for SQLiA, the proposed solution integrates detection, prioritization, and prevention. This may be used to supplement the current framework for sanitization and protection at the application and network layer.

Although numerous SQLi attack detection and prevention solutions have been developed in the literature, this research addresses gaps identified in current solutions (see Section 2, Section 4.4.3). It has been recognized there is an urgent need to develop a system that adopts the following features:

- **Multisource:** Many SQLi attacks target the application layer protocol, HTTP, leading most solutions to focus on HTTP traffic inspection [9,21]. However, as threat actors now exploit non-HTTP protocols such as DNS for data exfiltration and through various components, there is a need for a complete inspection of all elements of an HTTP request including the query parameters, HTTP headers and request body as well as network layer information. This resulted in multisource data analysis and evaluation of machine, deep and hybrid learning models for SQLIA detection.
- **Multiclass Classification:** Determining the intent and sophistication of the attack using a multiclass classification approach that classifies the complexity of the SQLIA (authentication bypass, classic SQLIA, blind SQLIA, remote code execution and DOS) against normal text and benign SQL operations. This allows for generalization to unseen attacks in the testing stage due to the volume and complexity of SQL injection techniques in practice.
- **Attack Prioritization:** Risk modelling through attack prioritization helps organizations allocate resources effectively as all SQLi attack types do not have equal severity or exploitability. By targeting the most critical SQLi threats, businesses can respond faster and save valuable time. This was achieved by streamlining vulnerability patching with CVSS, EPSS (Exploit Prediction Scoring system) and EE (Expected Exploitability) for increased efficiency in resource allocation.
- **Prevention:** This involves continuous monitoring techniques to prevent SQLi attacks by complementing detection functionality with preventive response measures to present an in-the-wild evaluation of the proposed scheme using real-time web and network traffic filtering.

To the best of our knowledge, this research proposes an original solution that can detect a wide variety of SQL injection types from multiple sources while providing alerting, active response and vulnerability prioritization capabilities that have not been demonstrated by existing research. Therefore, SQLiA detection is an unresolved problem incorporating these system requirements. The contribution of this research is the proposal of *SQLR34P3R*, a holistic solution which implements a fourfold approach to SQLi detection and prevention using the key features described above - *multi-source detection, multi-class attack classification, attack prioritization incorporating threat intelligence feeds and prevention through real-time continuous monitoring*. As of now, a solution with these features has not been proposed before in existing literature. We contribute to the research community efforts by building and sharing the code, dataset, and resources[1] for multisource and multiclass SQL injection attacks using machine learning techniques.

The rest of the paper is structured as follows. Section 2 presents an overview, evaluation, and gap analysis of existing solutions. Section 3 describes the project methodology for the classification and detection framework, risk analysis and threat modelling, as well as the datasets used, the model's hyperparameter settings, and the experiment design. Section 4 describes and discusses the results obtained in the experiment as well as the system design, implementation, deployment, and evaluation of the proposed solution. The challenges and limitations of the research are discussed in Section 5, while the conclusion and future research directions are provided in Section 6.

2. Literature review

This section describes some key findings in the literature by categorizing existing solutions under their thematic areas — traditional, machine learning and deep learning-based frameworks. The pros and cons of existing literature are described, as well as a gap analysis that locates the proposed research in the context of state-of-the-art SQLi attack detection and prevention.

[1] https://github.com/ap4ul/sql-injection-attack-detection-and-prevention

2.1. Traditional frameworks

Early SQLi detection frameworks adopted static analysis approaches like pattern matching to detect SQLi attacks. AMNESIA [22] was one of the earliest adopted SQLi detection and prevention solutions, which used hybrid analysis to detect SQLi. AMNESIA combines source code analysis with dynamic analysis operations, such as intercepting SQL queries during runtime to test for SQLi signatures. Contemporary static analysis approaches use regular expressions to detect and block SQLi traffic [23], which acts as a proxy to filter malicious SQLi traffic in real-time and authenticate users. Similarly, a framework which implements a combination of the BCRYPT hashing algorithm and the Aho–Corasick pattern-matching technique was employed in [24]. Aho–Corasick pattern matching technique stores a predefined set of patterns within a tree data structure, which the algorithm can traverse during the searching phase to find matches of malicious SQLi attack signatures. SAFELI [25], developed to detect SQLi vulnerabilities in ASP.NET programs, conducted decision-making through source code interpretation and considered all execution pathways without actual execution to determine possible SQLi attack signatures. The main disadvantage of proposed static analysis-based SQLi attack detection methods was their inability to detect unknown SQLi attacks [26].

2.2. Machine learning-based frameworks

In recent years, the popularity of using machine learning models to detect SQLi attacks has surged due to their effectiveness over traditional methods [26]. SQLIFIX [27] uses clustering to identify similarities between different PHP and Java codebases collected from GitHub that are vulnerable to SQLi and automatically generates remediation recommendations. Support Vector Machine (SVM) was used in conjunction with node centrality attributes, tokenisation and dimensionality reduction on the values following WHERE clauses in SQL statements for query component prioritization [28]. An alternative solution combined feature extraction methods such as query trees and Fisher score with an SVM classifier from the Waikato Environment for Knowledge Analysis (Weka) library [29]. The query trees for both benign and malicious SQLi data are generated using PostgreSQL. Feature extraction was performed by calculating the average and the standard deviation in Fisher Score to avoid repeated patterns. A multisource approach to SQLi attack detection was proposed, which captures application and network layer information from network traffic data using the *datiphy* appliance [30]. These two datasets are then combined to form a correlated dataset which is then used for classification by a range of machine learning classifiers from the Weka machine learning library. A predictive machine learning solution trained using SVM is deployed as a web proxy application programming interface (API) and can effectively handle high volumes of data [31]. SQLBlock, a plugin designed for PHP and MySQL applications, detects SQLi vulnerabilities through hybrid analysis preventing execution of unsafe PHP functions [32]. Several machine learning-based algorithms were investigated in a comparative analysis in [33] to detect SQL injection attacks at the network layer.

2.3. Deep learning-based frameworks

Neural networks such as Convolutional Neural Networks (CNN) offer increased scalability over traditional machine learning models and excel in detecting relevant text without requiring complex feature engineering [26]. Convolutional neural networks (CNNs) and Multilayer Perceptron (MLP) are two common deep learning models used in SQLi detection with some approaches performing decoding and tokenisation on the SQL statements through lexical analysis for extracting features [34]. Different research proposed a more traditional CNN-based approach to SQLi detection, which used a combination of request decoding and vectorization and outperformed the rule-based SQLi detector, ModSecurity [35]. Long short-term memory (LSTM), an alternative deep learning approach, reduces the number of false negatives that may arise due to manual feature extraction and addresses the overfitting problem by automatically generating SQLi attack payloads [36]. In a comparative analysis study performed against MLP, it was discovered that LSTM had better capabilities in detecting SQLi attacks as it does not require feature extraction and due to its proficiency in forming relationships between characters within the SQL statements [37]. A semantic learning-based detector known as synBERT was proposed, which uses embedding vectors to recognize the patterns within individual SQL queries which uses syntax tree structures to understand the grammatical context within SQL statements [26]. A recent proposal utilized the benefits of deep learning by employing a probabilistic neural network (PNN) to enhance SQLi attack detection accuracy [38]. The approach used 6,000 SQLi queries as malicious datasets and 3,500 benign queries alongside network traffic. A combination of regular expressions and tokenization was used for data pre-processing while Chi-Squared Test was used to extract relevant features from both SQL queries and network traffic.

2.4. State-of-the-art evaluation

The state-of-the-art SQLi detection and prevention solutions described above possess numerous advantages and limitations that have been presented in Table 1. None of the existing solutions for SQLi attack detection and prevention that were evaluated could classify detected SQLi attacks into their respective categories. They merely classified intercepted traffic as malicious or benign. Attack prioritization or risk modelling was not performed by any of the systems that were reviewed. Only 12% of the examined solutions utilized several sources to detect SQLi attacks, while 38% had active response or preventative features. By performing a gap analysis, critical limitations have been identified, highlighting areas for potential improvements. In this research, multiclass classification is not solely defined as a solution that can detect multiple types of SQLi attacks; it should also be able to attribute each detected SQLi attack to its corresponding type. It became evident that none of the evaluated state-of-the-art solutions in Table 1 could attribute detected SQLi attacks to their respective classes, even though some solutions were able to detect multiple types of SQLi attacks. Moreover, 50% of the research reviewed in [39] failed to specify the types of detected SQLi, highlighting the need for a multiclass classification-enabled solution. The absence of attack prioritization in the evaluated state-of-the-art solutions will make it challenging for businesses to allocate appropriate resources and may lead to delayed response to combat critical threats. Attack prioritization through risk modelling and threat intelligence refers to the process of assessing the security threats, risks and vulnerabilities based on their exploitability ease and likelihood. By integrating both risk modelling and threat intelligence into the solution businesses can create a comprehensive threat map and allocate resources effectively. Most of the solutions evaluated in Table 1 consider only the HTTP protocol as the attack vector, ignoring other potential SQLi attack vectors such as DNS. By implementing multi-source detection, the robustness of identifying these attacks can be enhanced, as it leverages various techniques and data sources, including non-HTTP protocols such as DNS. In this research, SQL Injection attacks are formulated as a multi-source detection (network traffic, HTTP headers, HTTP body and URL Parameters), multi-class classification (authentication bypass, classic SQLi, blind SQLi, remote code execution, and denial of service), prioritization and prevention problem. For a comprehensive overview of SQLi attack detection and prevention techniques, interested readers are referred to the systematic surveys [39,40].

Table 1
Summary and limitation of SQL injection attacks detection and prevention techniques.

Solution	Advantages	Disadvantages
AMNESIA [22]	- Efficient at reducing false positives. - No required to interact with the client or backend.	- Stored procedures attacks go undetected. - Specific to Java web apps.
Regular Expression pattern matching [23]	- Not required to interact with the client or backend.	- Stored procedures attacks go undetected. - Can be bypassed using advanced attacks.
Aho–Corasick pattern matching [24]	- Minimized false positives and negatives.	- Does not consider the overall SQL statement structure. - Can be bypassed using advanced attacks.
SAFELI [25]	- Interprets complex string conditions. - Analyses all potential execution flows.	- Only able to detect SQLi attacks in .NET applications. - The attack library needs to be compiled manually by security experts.
SQLIFIX [27]	- Language Agnostic. - Performs better than the Prepared statement replacement algorithm (PSR).	- Unable to detect SQLi in 139 code segments and certain conditions e.g., presence of some SQL modifiers, query errors, batch process support
SVM with node centrality [28]	- Low-performance overhead. - Safeguards many websites on the deployed server	- Solely focuses on WHERE clause values. - Lack of testing performed with diverse web apps.
SVM with Query Trees and Fisher Score [29]	- Distinguishes normal users from threat actors. - Eliminates redundant features.	- Limited SQL queries in the testing dataset. - Lack of detail on detectable SQLi attack types.
Multisource SQL Injection Detection [30]	- Accuracy comparable to advanced deep learning models with less overhead.	- Does not consider varied HTTP headers for inspection. - Lack of detail on detectable SQLi attack types.
Predictive machine learning using SVM [31]	- Handles large volumes of data efficiently.	- Only can be implemented with login endpoints. - Cannot detect sophisticated SQLi attacks [41].
SQLBlock Plugin [32]	- Effective against multiple content management platforms with no negative performance impact.	- Restricted to PHP web applications. - Does not detect the unsafe PHP function “eval()” as vulnerable.
CNN & MLP [34]	- Works well against classic SQLi attacks.	- Model not trained on advanced SQLi attacks.
Traditional CNN [35]	- Outperformed rule-based solution, ModSecurity, in terms of evaluation metrics.	- Trains models with limited types of SQLi attacks. - Not on par with SOTA in terms of performance.
LSTM [36]	- Mitigates overfitting and detects diverse SQLi attack types.	- Attack scenarios considered by the solution may not be comprehensive.
LSTM/MLP with advanced Feature Extraction [37]	- Enhances accuracy using a unique feature extraction implementation.	- Only evaluates input from URLs and overlooks alternative attack vectors.
synBERT [26]	- Offers higher flexibility and detection rate for previously unknown attacks.	- Model is trained on limited dataset.
PNN [38]	- High accuracy with low-performance overhead and false positive rate.	- Susceptible to interference from irrelevant features. - High complexity.

3. Methodology

This section presents the proposed methodology for the detection, classification, risk analysis and threat modelling of SQL injection attack (SQLIA). The implementation of the proposed solution, SQLR34P3R, was achieved through training the models using two distinct datasets: one mirroring real-world SQLi application layer attacks and the other representing SQLi attacks within network traffic, utilizing both traditional machine learning and deep learning models. The rest of the section describes the details of the different components of the proposed solution and the evaluation method.

3.1. Detection and classification framework

The overview of the detection and classification framework is presented in Fig. 2. In the first phase, suitable SQLi payloads and NetFlow datasets are collected and aggregated. In the second phase, the dataset is pre-processed and cleaned. The goal of this component is to improve the quality of the data and present it in a suitable format for the classification task. Next, features are extracted from the dataset for training and evaluating several machines and deep learning models. Finally, the method of evaluation of the SQLi detector model is presented. A detailed description of each phase is discussed in the subsequent sections.

3.1.1. Dataset collection

As shown in Table 2, the multiclass payload samples were gathered and combined from several open sources. Table 2 summarizes the payload and NetFlow dataset compilation’s source, data type, and sample count. Normal text and benign SQL operations make up the benign SQL payloads, with SQLMap [42] serving as the main data source. The categories of SQL injection techniques examined in this research are fully supported by SQLMap, an open-source penetration testing tool that automates the process of finding and exploiting SQL injection vulnerabilities and taking over database servers [42,43]. The SQLi payload dataset contains 37,709 benign samples and 32,314 malicious samples distributed into five classes of injection techniques. SQL injection attacks are included in the NetFlow datasets as malicious NetFlow data. To ensure generalization, the attacks are SQL injection for Union Query and Blind SQL injection using the SQLMap tool. DOROTHEA (Docker-based framework for gathering NetFlow traffic) [44] is used to produce the NetFlow traffic. The NetFlow dataset contains both benign and malicious traffic. The open repository [45] presents more information on the dataset’s implementation and infrastructure, while more information on the data processing is discussed in [33]. Approximately 50% of the traffic is malicious, while the rest is benign.

3.1.2. SQLi attack collection and labelling

For attack classes such as Authentication Bypass, Classic SQLi, Blind SQLi, and Remote Code Execution the corresponding SQLi payloads were already prelabelled by GitHub [46–48], SQLMap [42] and OWASP [49] respectively. The accuracy of these mappings and datasets from Kaggle [50] was further validated by performing a literature review where the characteristics of each malicious SQL query type were established. The literature reviewed for validating each attack characteristic is detailed in Table 3. For the NetFlow dataset [45], each traffic flow was pre-labelled as either malicious (1) or benign (0), reflecting a binary classification problem. Consequently, there was no

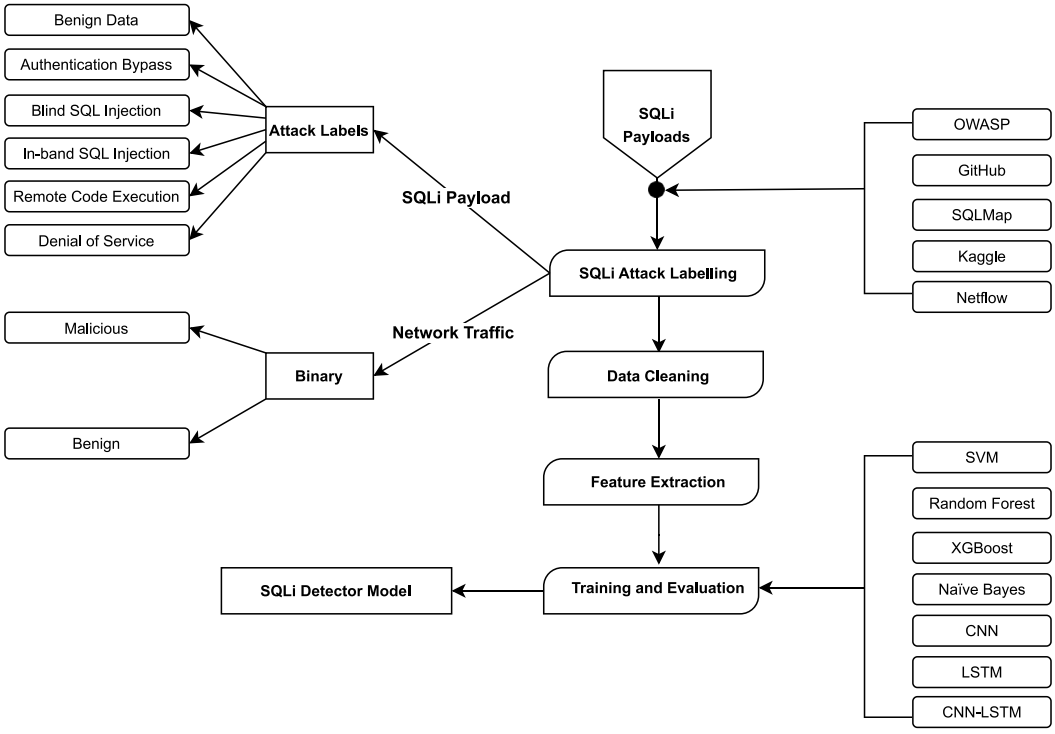


Fig. 2. SQLR34P3R detection and classification framework.

Table 2
Dataset distribution.

Category	# of Samples	Source
SQLi Authentication Bypass	247	GitHub [46–48]
Classic SQLi	15,474	SQLMap [42]
Blind SQLi	16,039	SQLMap [42]
Remote Code Execution	54	OWASP [49], Kaggle [50], SQLMap [42]
Benign (Normal Text & SQL Operations)	37,709	Kaggle [50]
Denial of Service	320	Kaggle [50]
NetFlow (Malicious & Benign)	457,233	Zenodo [45]

Table 3
SQLi Attack Labelling.

Payload characteristic (Malicious & Benign)	Attack type	Label	Lit. Source
alan' or '2'='2'– -	Authentication Bypass	0	[51]
; WAITFOR TIME '18:15:00'; – -	Blind SQL Injection	1	[51]
select derived_table.table1 from (select encode(decode(compress(convert(post) using latin1)),md5(concat(post,post))) as derived_table;	Denial of Service	2	[11]
' UNION SELECT 'a',NULL,NULL, NULL– -	Classic SQL Injection	3	[51]
; exec master..xp_cmdshell 'certutil -urlcache -f https://redteam/'–	Remote Code Execution	4	[52]
SELECT DEPARTMENT, COUNT(*) FROM EMPLOYEES;	Normal SQL Operation	5	[53]

need for further categorization. Table 3 showcases examples of each attack type along with its corresponding label and literature used to perform payload characterization.

3.1.3. Data pre-processing and data cleaning

Data pre-processing consists of standardizing data into a format suitable for the machine learning model. Any duplicate or empty entries that may have accumulated during dataset collection for the SQLi payload datasets were removed by executing bash scripts to improve dataset reliability. The NetFlow V5 dataset contains a total of 24 features, with their descriptions listed in Table 4. For the NetFlow dataset, the features ('exaddr', 'engine_type', 'engine_id', 'src_mask', 'dst_mask','src_as', 'dst_as', 'unix_secs', 'unix_nsecs', 'sysuptime', 'first', 'last', 'nexthop', 'srcaddr', 'dstaddr', 'input', 'output') were discarded due to having low variance, introducing bias and negative to no impact on the detection of malicious SQLi network traffic [33,54]. Detailed information on the dimensionality reductions of the NetFlow V5 features and data normalization of the features are well-documented in [33].

3.1.4. Feature extraction

Natural language processing (NLP) was performed on the SQLi payload dataset to convert the text-based SQLi payloads into machine-readable format. CountVectorizer and TF-IDFVectorizer are natural language processing techniques for vectorizing text, and have been used in literature to extract features from SQL payloads [55]. Term Frequency-Inverse Document Frequency (TF-IDF) was selected for conventional machine learning models due to its superior performance comparable to other feature extraction methods and its ability to understand essential signatures of SQLi attacks [56]. Term Frequency (TF) quantifies the presence of a particular word in individual SQLi payload data, while the Inverse Document Frequency (IDF) evaluates the whole payload dataset to identify terms that are infrequent attributing greater significance to those words. The formula for calculating TF-IDF is shown in Eq. (1).

$$\text{TF} - \text{IDF} = TF(w, P) \times IDF(w) \tag{1}$$

Table 4
Netflow V5 Features as described in [33].

Feature	Description
unix_secs	Time since 0000 UTC 1970 (s)
unix_nsecs	Remaining time since 0000 UTC 1970 (ns)
sysuptime	Current uptime (ms).
exaddr	IP address of flow exporter
dpkts	Number of packets in the flow
doctets	Total number of bytes in layer 3 of the packets within the flow
first	System uptime at the start of the flow
last	System uptime when the last packet in the flow was received.
engine_typ	Type of flow switching engine
engine_id	Flow slot number of the switching engine.
srcaddr	Source IP address
dstaddr	Destination IP address
nexthop	Next hop router IP address.
input	SNMP index of the input interface.
output	SNMP index of the output interface.
srcport	Source port number for TCP/UDP.
dstport	Destination port number for TCP/UDP.
prot	IP protocol type
tos	Type of service (ToS) for IP field
tcp_flags	TCP flags
src_mask	Number of prefix mask bits for the source address.
dst_mask	Number of prefix mask bits for the destination address.
src_as	Autonomous system number from either *source* or *pair*.
dst_as	Autonomous system number received by *destination* or *pair*

As shown in Eq. (1), TF is determined by evaluating the frequency of the word "w" within the payload "P". The IDF value is computed by dividing the overall count of "P" payloads in the dataset by the number of payloads that includes "w", as shown in Eq. (2).

$$\mathrm{IDF(w)} = \log \frac{P}{1 + p(w)} \tag{2}$$

Tokenization and Global Vectors for Word Representation (GloVe) are then applied to perform word embeddings for the CNN model to break down the SQLi payloads into tokens. GloVe is a word vectorizer technique that is used to contextualize and represent words in machine-readable format [57,58]. In contrast to conventional models such as Word2Vec that primarily analyse adjacent words within a given text, GloVe employs a more comprehensive approach by examining the co-occurrence frequency of words across the entire corpus. Word2Vec models typically focus on local context, using techniques like Continuous Bag of Words (CBOW) and Skip-gram to predict neighbouring words based on their proximity. This probabilistic framework captures local syntactic relationships effectively but may miss broader semantic contexts.

GloVe, on the other hand, constructs a global co-occurrence matrix, which captures the frequency of word pairs occurring together within a fixed context window throughout the entire corpus. This matrix is then factorized using a weighted least squares objective function, ensuring that the dot product of the resulting word vectors approximates the logarithm of the probability of the word pairs co-occurring. This method integrates global statistical information, enabling GloVe to capture both local and global semantic relationships, resulting in more nuanced and comprehensive word embeddings. For this research, we utilized the *glove.6B.100d* pre-trained word vectors. These vectors are 100-dimensional embeddings trained on a dataset of 6 billion tokens from Wikipedia and Gigaword. The 100-dimensional vectors provide a rich representation of words that balance detail and computational efficiency. Using these pre-trained vectors accelerates the research process by leveraging embeddings that have already been extensively validated and optimized, ensuring robust and accurate word representations. Furthermore, the GloVe embeddings' ability to capture both syntactic and semantic properties makes them particularly suitable for a variety of natural language processing (NLP) tasks, such as text classification, sentiment analysis, and information retrieval. By utilizing the *glove.6B.100d* vectors, our models benefit from high-quality, empirically validated embeddings that enhance the overall accuracy and robustness of our analyses. This comprehensive approach enables us to build sophisticated models capable of understanding and interpreting complex linguistic patterns inherent in large-scale textual data. The feature values from the NetFlow dataset were normalized to values between 0 and 1 to prevent any weighted bias due to some features having greater significance. Standard Scaler was used to perform data normalization due to its speed, high scalability, and linearity [59]. Standard Scaler, as a standardizing function, removes the mean and adjusts the variance of each feature to one so that all the feature values adhere to a uniform scale removing any scale-induced bias [33].

3.1.5. Classification & evaluation

The classification phase involves distinguishing between SQLi attack and benign traffic while also identifying the correct SQLi attack type if an attack is detected. For training the SQLi payload model, six separate dataset files are loaded into the program and labelled as mentioned in Table 3. After feature extraction, the combined dataset is divided with 60% designated for training data and 40% designated for testing data to achieve a more accurate evaluation of the model's performance on a diversified set of unseen data. The choice of a 60:40 split is to assess the trained classifier's performance with a larger test dataset, considering the limited dataset sizes for some classes, such as SQLi authentication bypass, remote code execution and denial of service (see Table 2). If the dataset is small, the 60/40 split provides a little more trustworthy test set [60]. Additionally, the samples were stratified due to the different number of individual datasets to prevent any bias arising from imbalanced datasets. The goal of stratification is to prevent the creation of unequal distributions of classes in the training and testing sets. The NetFlow dataset was used for binary classification. Both the provided NetFlow datasets were combined into a single data frame before being split for training and testing. The NetFlow datasets were split in a standard 70:30 ratio. In line with common practices for model generalization, a 70:30 train–test split ratio was selected for the NetFlow dataset for consistency as observed in current learning-based systems for web attack detection such as SQLi [61–63]. Empirical studies also demonstrate that the best performance is when 70%–80% of the data is used for training, and the remaining 20%–30% [64,65]. To evaluate the robustness of the proposed scheme, we designed experiments for both binary and multi-classification tasks SQLi attack detection using payloads and network traffic.

The machine learning and deep learning models chosen to perform the classification tasks were Support Vector Machines (SVM), Random Forests (RF), XGBoost, Naïve Bayes (NB), Convolutional Neural Networks (CNN) and long short-term memory (LSTM). Prior similar research demonstrated SVM, RF and NB to have strong performance in both binary and multi-classification tasks [28–31,66]. CNN and LSTMs have also been used for their efficiency, performance, and adaptability in both binary and multi-classification problems [34–37]. We also combined CNN and LSTM with GloVe embeddings for improved performance consistently across metrics, even with an unbalanced dataset [67]. While CNNs can effectively capture local semantics between texts, they struggle to understand the wider context, while LSTM models are time-consuming to train, which has resulted in the adoption of hybrid models to traditional CNN and LSTM in diverse domains of applications [68–72]. As a result, we also trained a hybrid CNN-LSTM model, which addresses these limitations by utilizing CNN for feature extraction and LSTM for wider contextual understanding between text data [73].

Several performance measures have been taken into consideration to assess the trained classification models. Since we have two different classification contexts — binary classification (NetFlow data) and multi-class classification (SQLi payloads), we define the dimensions of the confusion matrix for each context. For the SQLI payloads, the True Positives (TP) is the number of samples in each attack class correctly identified. The True Negatives (TN) represents the number of instances that were correctly classified as not being the class of interest for

each class. The False Positives (FP) is the number of samples that were incorrectly classified as the class of interest when they belong to a different class. The False Negatives (FN) represent the number of instances that were incorrectly classified as not being the class of interest when they do belong to that class. In a multi-class problem, there is one score for each class, counting any other class as a negative. In our case, there are 6 classes (Class 0 to Class 5) (see Table 3), and the status of an instance depends on the target class [74]. For example, for class 0 (Authentication Bypass SQLi): TP instances are class 0 predicted as class 0, FN instances are class 0 predicted as class $1, 2, 3, 4$ or 5, FP instances are class $1, 2, 3, 4$ or 5 predicted as class 0, and TN instances are class $1, 2, 3, 4$ or 5 predicted as class $1, 2, 3, 4$ or 5. For the NetFlow dataset, TP is the number of malicious flows accurately classified as such. TN refers to the number of benign flows correctly classified as benign traffic. FP is the number of benign flows incorrectly classified as malicious traffic, while the number of malicious flows incorrectly identified as benign traffic is reported by FN. We use precision, recall and F1-Score to measure the performance of the classifiers in binary and multiclass classification problems in SQLiA detection.

Precision is the ratio of true positives out of all positives. It is calculated as shown in Eq. (3). The precision for binary classification is calculated by dividing the total number of true positives and false positives by the number of true positives. The precision for multi-class classification is calculated by dividing the total number of true positives in all classes by the total number of false positives and true positives in all classes.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

Recall is the percentage of true positives out of all actual positives. It is calculated as shown in Eq. (4). Recall for binary classification is computed by dividing the total number of true positives and false negatives by the number of true positives. Recall for multi-class classification is computed by dividing the total number of true positives in all classes by the total number of false negatives and true positives in all classes.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4}$$

The F1-Score is the harmonic mean of precision and recall, and it is computed as shown in Eq. (5). In the multi-class classification context, the problem is evaluated as though it were a binary classification problem for every class individually. By following the same procedure for every class independently, we obtain a different F1 score for each class. After that, we calculate an overall performance metric using the macro F1-score. In the statistical analysis of machine-learning-based SQL injection detection systems, the F-Score is the most popular measure of predictive performance [30,33,36,37].

$$\begin{aligned} \text{F1 Score} &= \frac{2}{\frac{1}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}} \\ &= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned} \tag{5}$$

3.2. Risk analysis

Risk analysis was performed on the different SQLi attack classes to prioritize their risks. In this research, we aimed to model risks based on the identified SQLi classifications to aid efficient attack prioritization. The amalgamation of risk analysis and threat modelling revealed connections between detected SQLi attacks and known threat actors along with their attributes. The following frameworks were considered as a methodology for the risk analysis phase, as they are state-of-the-art solutions for vulnerability prioritization.

- **CVSS:** The Common Vulnerability Scoring System (CVSS) evaluates the severity of known vulnerabilities or Common Vulnerabilities and Exposures (CVEs) and is implemented by popular vulnerability management software [75,76]. However, using CVSS base score as a measure of risk has raised doubts as it only calculates the severity of vulnerabilities with no knowledge of the context surrounding the vulnerability [77]. Due to the obvious issues present in CVSS, many of the vulnerability prioritization solutions have stopped relying on CVSS while others such as Vulnerability Prioritization System (VPR) have combined both CVSS with ML models to provide additional context to assist risk assessment [78]. Although solutions such as VPR are better alternatives than relying solely on CVSS, these solutions lack transparency and cannot be adopted by businesses or organizations due to the closed-source nature of the system highlighting the need for open-source prioritization systems [79].
- **EPSS:** The implementation of the Exploit Prediction Scoring System (EPSS) addressed the limitations of CVSS by accurately forecasting the exploitability of vulnerabilities in the wild [80,81]. EPSS is known to be more effective than CVSS due to its increased threat awareness and ability to determine vulnerabilities being actively exploited. The main features of EPSS included calculating exploitability using public and private artifacts and its ability to create daily forecasts. The latest version of EPSS uses a 30-day prediction window. Although EPSS contains multiple benefits, it is noted that it is unable to be used as a primary risk metric due to the lack of vulnerability context, such as how the vulnerability is implemented in a specific system and what software the vulnerability affects [79].
- **EE:** Expected Exploitability (EE) [82] is a similar approach to EPSS in that the two approaches are used to determine the active exploitation of the vulnerability in the wild. EPSS is a machine learning scoring system, while EE is a deep learning-based system. Furthermore, both EE and EPSS recognize that a vulnerability's exploitation probability may vary over time or with the publication of new articles in connection with that vulnerability. EE keeps a record of known features about the disclosed vulnerability and assesses it against any new novel public artifacts that may be published after a period. EE determines the ease of exploitation and probability of an exploit being developed for that vulnerability.
- **RRM:** The OWASP Risk Rating Methodology (RRM) uses two main components to calculate the severity: likelihood and impact. The "Likelihood" component is made up of factors such as threat agent and vulnerability information while the "Impact" component consists of technical and business impacts [83]. Although the framework offers a straightforward and efficient approach to risk analysis, a significant drawback of using OWASP RRM is its reliance on experts to assign values for each factor which can lead to human error [84].
- **CAVP:** The Context-Aware Vulnerability Prioritization (CAVP) [85] is a recent solution that prioritizes risks based on an organization's assets. Moreover, CAVP verifies the vulnerabilities with trusted sources and performs environmental scanning to identify the CVEs that will affect the organization's technological environment. CAVP supplements the original CVSS approach by combining temporal metrics with expert verification and heuristic rule-based methods. However, CAVP's implemented metrics lack validation and CAVP has only been compared with CVSS, not with other solutions like EPSS or EE [85].

This research proposes, RISK R34P3R, a vulnerability prioritization model that combines CVSS, EPSS, EE and OWASP RRM risk calculation formula that considers exploitability, exploit availability and severity metrics. Solutions such as VPR and CAVP were excluded due to the closed-source properties [79]. CVSS's transparency, backed by security expert input makes it a reliable choice to determine vulnerability severity. Although EPSS use CVSS metrics as a data source, EPSS does not calculate severity scores but focuses on the likelihood of exploitation of the vulnerability [86]. Moreover, EPSS and EE have distinct features

Table 5
OWASP RRM overall risk rating [83].

	Overall risk severity			
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Informational	Low	Medium
		LOW	MEDIUM	HIGH
		Likelihood		

which set them apart from each other, such as the use of public data sources in EE, and the use of private and public data sources in EPSS [81,82,86]. Additionally, EPSS is concerned with predicting the likelihood of vulnerability exploitability in the wild whereas EE is more focused on assessing exploitation difficulty using only public data artifacts. Frameworks such as CVSS, EPSS and EE focus on vulnerabilities within known CVE identifiers. For a comprehensive overview of data-driven software vulnerability assessment and prioritization, the interested reader is referred to [87].

To enable risk-induced vulnerability prioritization of SQLi attack classes, CVEs were collected from the National Vulnerability Database (NVD) spanning the years March 2020–June 2023. For categorization, keywords such as "Authentication Bypass", "Blind", "Classic", "In-Band", "Denial of Service" and "Remote Code Execution" were used to retrieve relevant CVEs for each class of SQLi. The risk severity was calculated using the risk calculation formula specified by OWASP RRM [83] by combining the CVSS, EPSS and EE metrics as shown in Eq. (6). The proposed risk formula is also consistent with the industry definition of contextual prioritization and risk-based prioritization, where the probability of exploitation, impact and severity are key components [88].

$$\text{Risk Severity} = AVG(N(EPSS)N(EE)) \times N(CVSS) \tag{6}$$

CVSS will be considered as an impact measure as CVSS measures how the vulnerability impacts the confidentiality, integrity, and availability of data [79]. Both EPSS and EE will form the likelihood measure as they both calculate the likelihood of vulnerabilities being exploited [81,82]. EPSS and EE generate outputs as values between 0 and 1, so the output values will need to be normalized using the Min-Max formula [89] as shown in Eq. (7). The CVSS is also normalized along with EPSS and EE using the same formula.

$$NV = \frac{V - \min}{\max - \min} \times 9 \tag{7}$$

To determine the severity of the risk, both the likelihood metrics (EPSS, EE) and impact metrics (CVSS) were calculated separately and were given an individual level based on OWASP scaling, where 0 to < 3, 3 to < 6 AND 6 to 9 represents LOW, MEDIUM, and HIGH for likelihood and impact levels [83]. After determining the likelihood and impact rating, the overall risk severity rating can be computed using the overall risk rating table from OWASP RRM as shown in Table 5. To further demonstrate that the proposed risk scoring was grounded in sound methodology, a similar proposal was made in [90] for streamlining vulnerability patching by combining CVSS, EPSS, and CISA KEV [91], thereby categorizing vulnerabilities into five priority levels for efficient resource allocation.

3.3. Threat modelling

Since vulnerability management on its own does not offer a consistent understanding of how adversaries utilize vulnerabilities to further their objectives, it is challenging to prioritize vulnerabilities effectively without the context provided by integrating vulnerability and threat information [92]. To bridge vulnerability management and threat management, there have been calls for threat-based vulnerability management which has led to the mapping of vulnerabilities with ATT&CK techniques by industry practitioners [93] and academic researchers [94–96]. However, most approaches for risk-based prioritization, such as EPSS, EE, etc., focus on factors related to the vulnerability, with no consideration for factors related to the threat agent or actor.

To provide a comprehensive approach using threat-based vulnerability management, we combine the proposed risk analysis process with intelligence gathering. This research recognizes the need for a more comprehensive approach to understanding vulnerabilities by providing a view of the threat landscape to perform informed and decisive action. The following threat intelligence feeds were integrated.

- **AbuseIPDB** [97]: This is an online repository linked to malicious activities. AbuseIPDB provides information such as the geolocation information of reported IP addresses, associated hostnames, Fully Qualified Domain Names (FDQNs), and any reported malicious activities performed from the queried IP addresses [98].
- **AlienVault Open Threat Exchange (OTX)** [99]: AlienVault OTX gathers intelligence feeds through "pulses". Each pulse contains threat information, associated techniques and indicators of compromise (IoCs) like URLs, hashes, and hostnames. For this project, the malware families and the TTPs of the threat actor associated with the detected malicious source of the attack are extracted [100].
- **Shodan** [101]: Shodan scans and indexes information related to Internet-facing devices such as the operating system, running services and vulnerabilities. By monitoring the attacker's machine over time, it is possible to gain insights into their behaviour and tactics [102].

4. Evaluation

Through an in-depth review of the literature, the following research questions (RQs) were established to underpin the analysis of existing gaps in state-of-the-art SQLi detection and prevention solutions.

- **RQ1. How effective are multi-source SQL injection detection methods in mitigating SQL injection attacks?**: This research evaluated the effectiveness of multi-source SQLi detection methods compared to traditional single-source solutions. The project will detect SQLi attacks from multiple sources like HTTP headers, parameters, and non-standard protocols such as DNS to cover the different attack scenarios specified.
- **RQ2. What are the advantages of using multi-classification-based machine learning models to detect and categorize detected SQL injection payloads?**: Training a machine learning model with multi-classification capabilities can attribute each detected SQLi attack to its specific type. This assists in developing a risk model that assesses the threat and impact of each attack variant.
- **RQ3. How has the proposed risk analysis enhanced state-of-the-art vulnerability prioritization systems?**: The research aims to determine the risk of individual attack types using risk modelling to quantify the potential impacts of SQLi attacks and propose a novel risk modelling approach that enhances current exploit prediction approaches. Additionally, the research aims to align risk modelling with threat intelligence to gain a comprehensive understanding of the threat landscape.
- **RQ4. How can continuous monitoring techniques prevent SQLi attacks?**: This research explores the possibility of complementing detection functionality with preventative response measures, including IP blocking and request screening, to prevent malicious requests from reaching the backend DBMS systems.

The proposed scheme's SQLi attack detection and classification are covered in RQ1 and RQ2, while the SQLi vulnerability prioritization utilizing CVEs and threat information is covered in RQ3, and the proposed scheme's SQLi attack prevention is covered in RQ4.

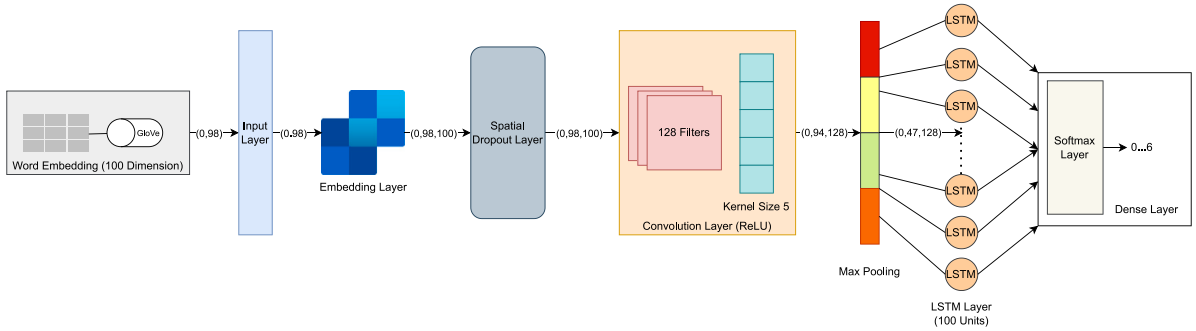


Fig. 3. The hybrid CNN-LSTM architecture for SQLi attack detection in HTTP payloads.

4.1. RQ1 - multisource SQLi attack detection

4.1.1. Experimental design

The machine learning and deep learning models were evaluated based on their ability to accurately identify and differentiate between the SQLi injection attacks and benign traffic. The top-performing models for both SQLi payload and NetFlow data were selected as the primary models for the respective classification tasks. GridSearchCV was employed for hyperparameter tuning to identify the optimal parameter combinations for detecting SQLi attacks. After the tuning process, the recommended parameters for each machine-learning model are as follows:

- **SVM**: The regularization parameter was configured to "10", using a "linear" kernel type and a kernel coefficient of "1.0".
- **Random Forest (RF)**: Hyperparameter tuning had no beneficial impact on the default settings.
- **XGBoost**: The objective was configured as "multi:softmax" due to the multiclass problem, and the tree's maximum depth was established at "8".
- **Naïve Bayes (NB)**: Hyperparameter tuning had no beneficial impact on the default settings.
- **Deep Learning Models**: The Sparse Categorical Crossentropy was employed as the loss function using the Adam optimizer. With a batch size of 54, 10 epochs were found to be optimal.

For the three deep learning model implementations, GloVe word embeddings are used to understand word semantics. For the CNN implementation, the 1D convolutional layer is combined with the max pooling layer to determine key textual patterns and filter the most important features. A dense layer with 64 nodes is added to process the filtered features and a dropout layer is used to prevent overfitting. LSTM architecture utilizes a 100-unit LSTM layer to contextualize the order of words. The dropout layer is then used to prevent overfitting while a dense layer with 64 nodes is then used to process the features. For the combined CNN and LSTM model shown in Fig. 3, the CNN is used for feature extraction which identifies and extracts short patterns from text. The max pooling layer is then used to highlight key patterns and reduce the data size. Finally, the LSTM layer is implemented to understand the long-term dependencies within text sequences to gain a wider contextual understanding of text data. The final layer employs softmax activation for the three architectures to provide a probability distribution over the six potential classes.

The following parameters were recommended after hyperparameter tuning the models used for the Netflow SQLi attack detection:

- **SVM**: The regularization parameter was set to "1.0", using a "squared hinge" loss and an L2 regularization penalty.
- **Random Forest (RF)**: 80 decision trees were combined with a minimum sample split of 0.1.
- **XGBoost**: 100 decision trees were combined with a learning rate and L1 regularization of 0.1.

Table 6
Overall Precision, Recall and F1-Score obtained using the Multiclass SQLi Attack.

Algorithm	Precision	Recall	F1-Score
RF	99.0488%	94.0495%	96.2696%
SVM	99.1009%	93.7242%	96.1164%
XGBoost	99.3310%	92.8757%	95.6706%
NB	96.5961%	79.0139%	85.2463%
CNN	96.9742%	84.5254%	88.8691%
LSTM	**99.5635%**	**94.2610%**	**96.6700%**
CNN+LSTM	**97.6631%**	**95.1138%**	**96.3004%**

- **Naïve Bayes (NB)**: Hyperparameter tuning had no beneficial impact on the default settings.
- **Deep Learning Models**: The Binary Crossentropy was employed as the loss function using the Adam optimizer. With a batch size of 32, 20 epochs were found to be optimal.

Three distinct deep-learning architectures were evaluated for the classification of the NetFlow dataset. The one-dimensional CNN model implements a convolutional layer featuring 64 filters and the "relu" activation function. This is followed by dimensionality reduction through max pooling with a pool size of 2. The features are then flattened to transition from the convolutional to the dense layer with sigmoid activation. The LSTM architecture implements a 100-unit LSTM layer with both dropout and recurrent dropout rates set at 0.2 to reduce overfitting. The LSTM layer then connects to a Dense layer with sigmoid activation. The hybrid CNN-LSTM model, as shown in Fig. 4, takes advantage of both models by combining the 100-unit LSTM layer with the convolutional and max pooling feature extraction technique of the CNN before connecting to the dense layer with sigmoid activation for binary classification which is consistent for the three architectures.

4.1.2. Results

RQ1 aims to establish the benefits of multi-source data analysis and evaluation of traditional and hybrid learning techniques for SQL injection attack detection. To answer RQ1, the performance of the learning-based framework was evaluated on the two different data sources. Table 6 shows the overall performance of the models in detecting SQLi payloads. The results from the classifier evaluation presented in Table 6 highlight that the deep learning models outperformed the majority of the traditional machine learning models for the multiclass SQLi payloads classification. Even though both Random Forest and XGBoost outperformed the traditional CNN model. The LSTM model performs slightly better than the hybrid CNN+LSTM model due to its capability to handle imbalanced datasets effectively, even though the CNN-LSTM had a better recall in identifying all the classes accurately for SQLi payload detection. This observation aligns with findings in existing literature where LSTM demonstrated better performance compared to the CNN-LSTM model in some scenarios [67].

The results of the machine learning models' evaluation for the NetFlow data are presented in Table 7, along with a detailed analysis of

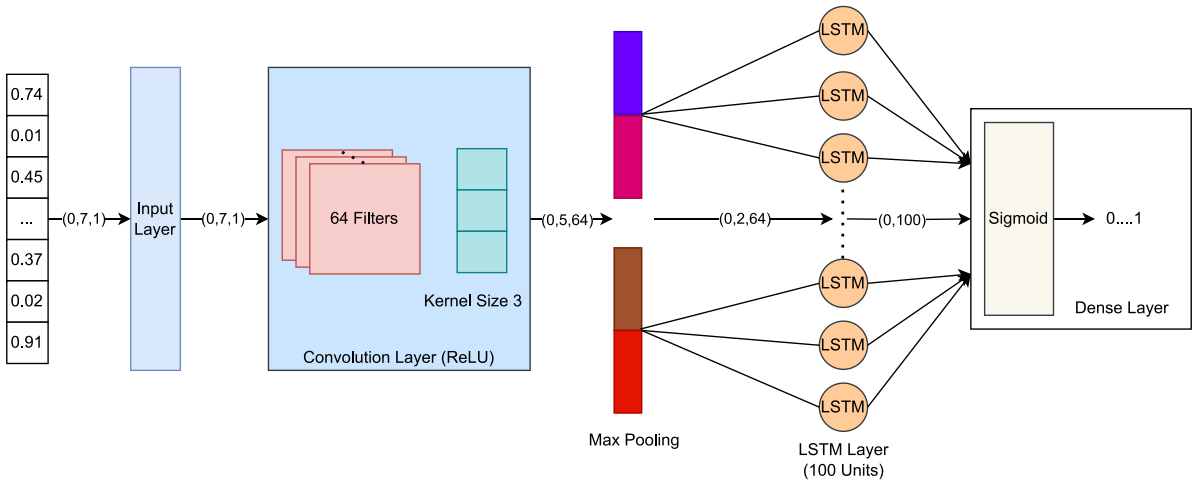


Fig. 4. The hybrid CNN-LSTM architecture for SQLi attack detection in flow data.

Table 7
Precision, Recall and F1-Score obtained using the NetFlow dataset [45] in comparison with SOTA which is a Logistic Regression-based model (LR) [33] on SQLi attack detection in NetFlow data.

Algorithm	Class	Precision	Recall	F1-Score
RF	Malicious	78.78%	99.86%	88.08%
	Benign	99.81%	73.06%	84.37%
	Average	89.295%	86.4633%	86.2228%
SVM	Malicious	90.16%	99.86%	94.76%
	Benign	99.85%	89.09%	94.16%
	Average	95.0026%	94.4740	94.4612%
XGBoost	Malicious	66.12%	99.87%	79.56%
	Benign	99.74%	48.75%	65.49%
	Average	82.9269%	74.3103%	72.5253%
NB	Malicious	92.91%	97.16%	94.99%
	Benign	97.02%	92.58%	94.75%
	Average	94.9645%	94.8676%	94.8663%
CNN	Malicious	92.39%	99.88%	95.99%
	Benign	99.87%	91.77%	95.64%
	Average	96.1288%	95.8207%	95.8163%
LSTM	Malicious	98.88%	23.39%	37.83%
	Benign	56.53%	99.73%	72.15%
	Average	77.7008%	61.5609%	54.9907%
CNN+LSTM	**Malicious**	**96.64%**	**98.72%**	**97.67%**
	Benign	**98.69%**	**96.56%**	**97.61%**
	Average	**97.6644%**	**97.6409%**	**97.6413%**
LR [33]	Malicious	94.90%	99.90%	97.40%
	Benign	99.90%	94.60%	97.20%
	Average	**97.4%**	**97.3%**	**97.30%**

the model performance for each class and are contrasted with the state-of-the-art approach [33] that directly addresses SQLIA detection in flow data. For the NetFlow classification, the hybrid CNN-LSTM model outperformed the other ML and DL models. The balanced performance of the model across both classes suggests its ability to adapt and deliver consistently high results regardless of the class distribution. While other classifiers such as SVM, Naïve Bayes and CNN showed good results, the CNN-LSTM hybrid model's consistently high performance across both overall and class-specific evaluations makes it the most reliable model for the NetFlow dataset.

We compared our models with the state-of-the-art SQL injection attack detection in network flow data in [33] since we used the same dataset [45] created and used by the authors. After evaluating several machine learning-based algorithms such as K-Nearest Neighbours (KNN), Logistic Regression (LR), Linear Support Vector Classification (LSVC), Perceptron with stochastic gradient descent (SGD) and Random Forest (RF), an average F1-score of 97.3% was obtained with a Logistic Regression-based model. As can be seen in Table 7, the proposed CNN+LSTM model demonstrated better results with an average F1-score of approximately 98%, while also having better precision and recall rates. The difference between the results obtained from the CNN+LSTM and the LR model lies in the complexity of the model. Logistic regression is a simple linear model that is easy to understand and interpret. On the other hand, deep learning utilizes complex neural networks that can identify intricate patterns in large datasets. We have demonstrated that hybrid deep learning algorithms are a better fit for detecting SQLi in flow data.

This research demonstrates that detecting SQL injection attacks on web traffic using a multiclass classification approach is achievable with traditional machine learning, deep learning, and hybrid models. The results of the multiclass approach imply that it is possible to detect malicious payloads from benign SQL payloads, which are text and normal SQL operations and classify the SQLi in their respective classes. From a multi-source data analysis perspective, we can conclude that deep learning algorithms are better for SQLIA attack detection and classification, as shown in Tables 6 and 7. While LSTM marginally performed better than the proposed hybrid CNN-LSTM model on the payloads' dataset by 0.38%, the hybrid model outperformed flow data by 43.68%. This finding implies that for multi-source data analysis in SQLIA detection and classification, hybrid deep learning models are preferable to traditional deep learning models. The exemplary performance of hybrid models over traditional models is consistent with other domains of application such as anomalous query access control [69], malware homology analysis [103], automated atrial fibrillation detection [70], movies review sentiment analysis [71] and short-term energy consumption prediction problem of the individual household [72]. In this research, we have collected data from two points — payloads and network flow data and have thus provided multi-source data analysis and evaluation of machine and deep learning techniques for SQL Injection Detection.

4.2. RQ2 - multiclass SQLi attack detection

RQ2 aims to investigate the advantages of formulating SQL injection attack detection as a multiclass classification problem. To answer RQ2, a detailed breakdown of the classifiers' performance for each class of SQLi attack and benign payloads is presented in Table 8. An interesting observation is that all the 7 learning algorithms investigated had a low F1-score (< 90%) for the remote code execution class compared to the other classes, which means that some data points on the test set could not be matched. Also, CNN and NB performed poorly on the denial-of-service class with a low F1-score of < 60%, while returning high precision for the class. These findings suggest that a multiclass

Table 8
Precision, Recall and F1-Score obtained for each attack class using the SQLi Payloads dataset.

Algorithm	Class	Precision	Recall	F1-Score
RF	Authentication Bypass (0)	94.68%	89.90%	92.23%
	Blind SQLi (1)	99.75%	99.94%	99.84%
	Denial of Service (2)	100.00%	98.44%	99.21%
	Classic SQLi (3)	100.00%	99.95%	99.98%
	Remote Code Execution (4)	100.00%	76.19%	86.49%
	Benign payloads (5)	99.86%	99.88%	99.87%
SVM	Authentication Bypass (0)	95.60%	87.88%	91.58%
	Blind SQLi (1)	99.81%	98.47%	99.14%
	Denial of Service (2)	100.00%	100.00%	100.00%
	Classic SQLi (3)	99.97%	99.90%	99.94%
	Remote Code Execution (4)	100.00%	76.19%	86.49%
	Benign payloads (5)	99.22%	99.90%	99.56%
XGBoost	Authentication Bypass (0)	96.67%	87.88%	92.06%
	Blind SQLi (1)	99.56%	99.77%	99.67%
	Denial of Service (2)	100.00%	98.44%	99.21%
	Classic SQLi (3)	100.00%	99.94%	99.97%
	Remote Code Execution (4)	100.00%	71.43%	83.33%
	Benign payloads (5)	98.75%	99.81%	99.78%
NB	Authentication Bypass (0)	96.34%	79.80%	87.29%
	Blind SQLi (1)	95.99%	95.18%	95.59%
	Denial of Service (2)	94.73%	42.19%	58.38%
	Classic SQLi (3)	95.03%	95.98%	95.50%
	Remote Code Execution (4)	100.00%	61.90%	76.47%
	Benign payloads (5)	97.48%	99.03%	98.25%
CNN	Authentication Bypass (0)	90.43%	85.86%	88.08%
	Blind SQLi (1)	98.68%	99.86%	99.26%
	Denial of Service (2)	92.86%	40.63%	56.52%
	Classic SQLi (3)	100.00%	99.90%	99.95%
	Remote Code Execution (4)	100.00%	80.95%	89.47%
	Benign payloads (5)	99.89%	99.95%	99.92%
LSTM	Authentication Bypass (0)	97.67%	84.85%	90.81%
	Blind SQLi (1)	99.94%	99.86%	99.90%
	Denial of Service (2)	100.00%	100.00%	100.00%
	Classic SQLi (3)	99.97%	99.95%	99.96%
	Remote Code Execution (4)	100.00%	80.95%	89.47%
	Benign payloads (5)	99.80%	99.95%	99.88%
CNN + LSTM	Authentication Bypass (0)	91.75%	89.90%	90.82%
	Blind SQLi (1)	99.89%	99.95%	99.92%
	Denial of Service (2)	100.00%	100.00%	100.00%
	Classic SQLi (3)	99.98%	99.95%	99.97%
	Remote Code Execution (4)	94.44%	80.95%	87.18%
	Benign payloads (5)	99.91%	99.93%	99.92%

classification approach improves the generalizability and robustness of detection models for SQLi.

In a systematic review of 60 research papers on SQL injection attacks detection and prevention techniques conducted in [39], 50% of the papers focused on general attacks, 7% on order of injection, 3% on server, 2% on in-bound/out-bound, while 38% focused on attacks of both 1st order and error-based types. This shows that the SQLiA attacks classification in the state-of-the-art would not be generalizable to unseen attacks such as the one considered in this study. For instance, in the attack classification, none of the papers considered remote code execution (RCE) and denial of service (DOS). This suggests that leveraging an SQLi vulnerability on a web application to perform RCE on the server hosting the target application would not be detected, even though this class of attack has been exploited in CVE-2021-42258 to target companies, such as BillQuick Web Suite [4]. Also, Denial of Service through SQLi, which is caused by performing recursive SQL operations to overload the database servers, rendering it unavailable and preventing users from performing database operations, was not classified by existing solutions.

As mentioned above in Section 2 and Table 11, no research formulated the SQLIA detection in web traffic as a multi-class classification problem. As a result of formulating the SQL injection detection problem as a binary classification, existing methods suffer from generalization to unseen attacks in the testing stage. A binary classification of benign and malicious SQLi payloads assumes the same or similar distributions between training and test sets for both classes. However, this is difficult because of the volume of SQL injection techniques in practice. For instance, six SQL injection techniques are supported by SQLMap, namely *boolean-based blind*, *time-based blind*, *error-based*, *UNION query-based*, *stacked queries* and *out-of-band* [42]. The majority of the SQLi attacks occur through the URL and request body parameters of GET and POST requests, respectively. However, threat actors have started to exploit alternative web application parameters such as HTTP header fields (including Cookies, X-Forwarded-For, User-agent and Referer) [9]. The emergence of various SQLi detection and prevention solutions has prompted

threat actors to adapt their strategies, leading to the execution of complex SQLi attacks and advanced evasion techniques that combine SQL Injection with DDOS attacks, DNS Hijacking, XSS, cross-domain policies of Rich Internet application and insufficient authentication [11–13]. This theory has been advanced in the context of voice spoofing detection, where binary classification approaches of bonafide and synthetic speech suffer from generalization due to unseen spoofing attacks in the testing stage [104–106]. By formulating the SQL Injection attack problem as a multi-class classification approach, we have demonstrated the generalizability of the approach.

4.3. RQ3 - SQLi vulnerability prioritization

To address RQ3, we collected CVEs from the NVD spanning from March 2020 to June 2023 along with their corresponding CVSS. We used the EPSS and EE API client to generate the scores using the identifiers of each CVE. Finally, we combined the CVSS, EPSS and EE scores to produce the RISK R34P3R risk ratings. Each business has a different capacity to address application vulnerabilities, with resource availability playing a key role. Smaller companies with limited budgets and resources might prioritize less "effort" by choosing to address the most critical vulnerabilities without achieving complete *coverage* of all the vulnerabilities. For the proposed SQL injection vulnerability management in this research, we evaluated it using the performance metrics of *coverage* and *additional effort*, which are the metrics that matter for the performance of vulnerability management [80,81,107]. *Coverage* refers to the number of vulnerabilities identified for patching across the various SQLi attack classes. Meanwhile, *additional effort* is defined as the number of vulnerabilities that do not need an urgent fix but are within the "coverage" scope to be patched. *Additional effort* is estimated by examining vulnerabilities that have at least two metrics below a given threshold, while the evaluated metric is above the threshold.

CVSS recommends remediating vulnerabilities with base scores ≥ 7, whereas EPSS suggests remediating vulnerabilities that score ≥ 0.022. Consider CVE-2023-36284, an SQLi authentication bypass attack with a CVSS score of 7.5, EPSS score of 0.000760000 and EE score of 0.0109. While the CVSS score is over the CVSS remediation threshold of 7.0, both EPSS and EE scores fall under their 0.022 and 0.50 thresholds [81, 82]. CVSS identifies this vulnerability as high severity, however, the vulnerability is not actively exploited (EPSS) or is not easily exploitable (EE) decreasing the need to prioritize it and increasing the additional effort required for patching if the vulnerability falls under the coverage scope. Table 9 provides the coverage and additional effort required to remediate the vulnerability for each attack class using CVSS, EPSS, EE and RISK R34P3R.

According to Table 9, RISK R34P3R stands out as the most effective vulnerability prioritization and exploit prediction system across the different SQLi attack classes. While CVSS offers high coverage, it also demands significant additional effort with 88.89%. EPSS, despite its precision in identifying the most urgent vulnerabilities, provides limited coverage as low as 6.90% for authentication bypass. Moreover,

Table 9
Vulnerability coverage and additional effort.

Authentication bypass		
	Coverage	Additional Effort
CVSS	96.55%	29.31%
EPSS	6.90%	0.00%
EE	67.24%	3.45%
Risk R34P3R	65.52%	0.00%
Blind SLQ		
CVSS	88.89%	42.22%
EPSS	5.19%	0.00%
EE	53.33%	6.67%
Risk R34P3R	47.41%	0.74%
Classic SQLi		
CVSS	80.00%	32.00%
EPSS	4.00%	0.00%
EE	52.00%	4.00%
Risk R34P3R	48.00%	0.00%
Denial-of-Service SQLi		
CVSS	100.00%	88.89%
EPSS	0.00%	0.00%
EE	11.11%	0.00%
Risk R34P3R	22.22%	11.11%
Remote Code Execution SQLi		
CVSS	100.00%	57.78%
EPSS	7.78%	0.00%
EE	41.11%	0.00%
Risk R34P3R	41.11%	0.00%

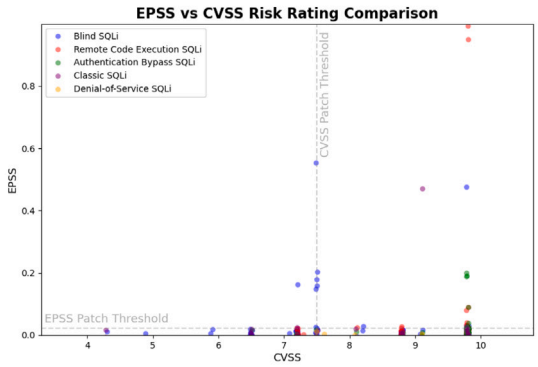
Table 10
Average risk ratings for each SQLi class using CVSS, EPSS, EE & RISK R34P3R.

SQLi Class	#	CVSS	EPSS	EE	Ours
Authentication Bypass	58	Critical	0.015158	0.663932	High
Blind	135	High	0.013951	0.543993	Medium
Denial-of-Service	25	High	0.001098	0.112075	Medium
Classic	9	High	0.021313	0.521872	Medium
Remote Code Execution	90	Critical	0.027397	0.418329	Medium

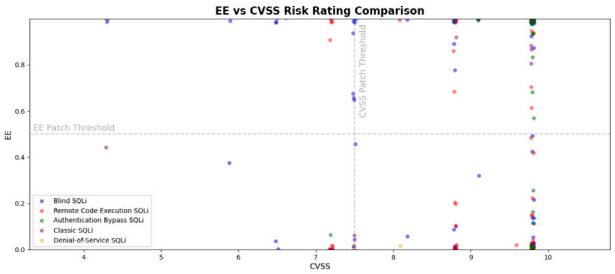
even though EE provides better coverage with 67.24% for authentication bypass, it still requires additional efforts with 6.67% for blind SQLi. In contrast, RISK R34P3R consistently demonstrates a balance, with coverage of 65.52% with no additional efforts for authentication bypass SQLi exploits. This indicates that the combination of CVSS, EPSS and EE by RISK R34P3R, ensures that the final risk rating not only considers the inherent severity of the vulnerability but also the likelihood of exploitability and the existence of published exploits helping the business focus on patching vulnerabilities that are actively being exploited that can cause critical impact.

Table 10 presents the average CVSS, EPSS, EE and RISK R34P3R scores for each class. The RISK R34P3R score for each class was calculated using the proposed risk severity formula in Eq. (6), the normalization metric in Eq. (7) and the overall risk rating presented in Table 5. A detailed methodology of the risk analysis is presented in Section 3.2 Both the "Authentication Bypass" and "Remote Code Execution" SQLi classes have a CVSS severity of "Critical". However, both attack classes present a relatively low EPSS rating. The "Authentication Bypass" class possess a high EE score suggesting a higher likelihood of existing exploits for this SQLi attack class. In contrast, "Remote Code Execution SQLi" has a lower EE score. Consequently, the RISK R34P3R model assigns them risk severities of "High" and "Medium" respectively after combining the CVSS, EPSS and EE scores.

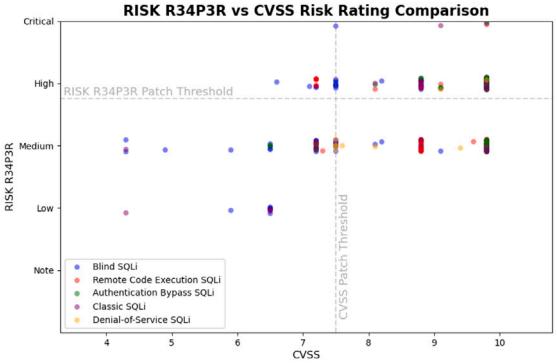
In Fig. 5, we provide a vulnerability prioritization visualization that compares the four risk metrics investigated in this study for each class of SQLi attack. Figs. 5(a), 5(b), and 5(c) illustrate the relationships



(a) EPSS vs CVSS Risk Rating Comparison



(b) EE vs CVSS Risk Rating Comparison



(c) RISK R34P3R vs CVSS Risk Rating Comparison

Fig. 5. Vulnerability prioritization visualization of SQLiA.

between EPSS (Exploitation Prediction Scoring System), EE (Expected Exploitability), and the proposed RISK R34P3R framework, in comparison to CVSS (Common Vulnerability Scoring System) scores. These figures are based on 317 common vulnerabilities, collected from March 2020 to June 2023. To provide some context, EPSS and EE generate prediction scores on a scale from 0 to 1 (or 0% to 100%), where higher scores indicate a greater likelihood of exploitation or ease of exploitation. On the other hand, RISK R34P3R assigns severity ratings, ranging from "Note", "Low", "Medium", "High", to "Critical". Figs. 5(a) and 5(b), reveal that a substantial majority of vulnerabilities are clustered towards the lower end of the plots. CVSS recommends patching for 259 vulnerabilities, whereas only 19 vulnerabilities have EPSS scores exceeding the remediation threshold. This observation suggests that relying solely on CVSS for vulnerability prioritization may prove inefficient, as it recommends the patching of vulnerabilities that are not easily exploitable. Moreover, relying exclusively on EPSS falls short in terms of coverage, as it leaves 298 vulnerabilities unaddressed, which does not bode well for the overall security posture of the business.

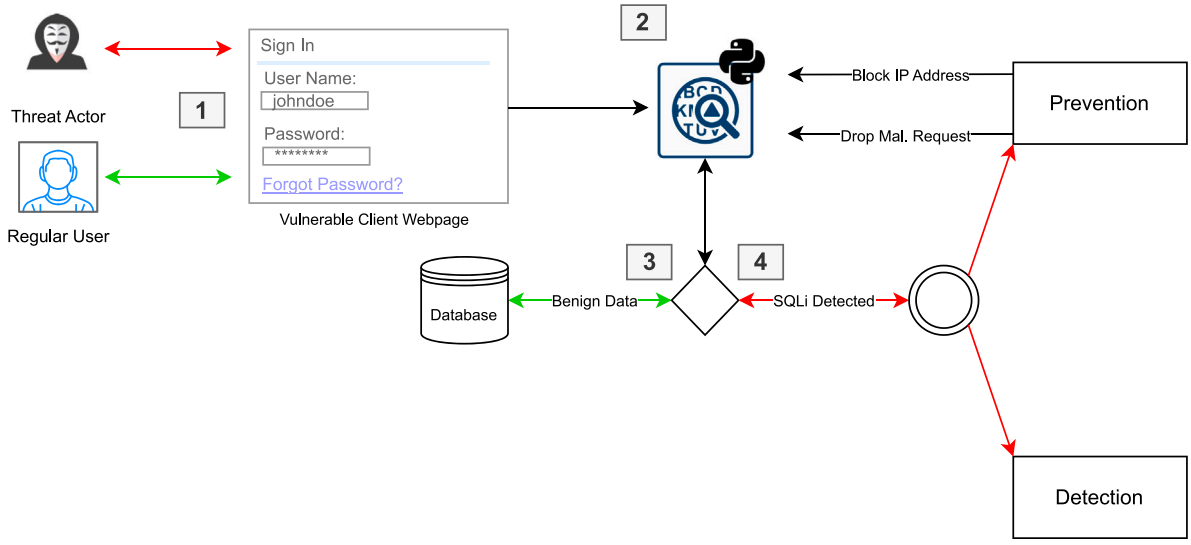


Fig. 6. SQLR34P3R: Process flow.

Similarly, in Fig. 5(b), the analysis of EE shows that 217 vulnerabilities exceed the EE threshold, which is lower in number than the vulnerabilities within the CVSS remediation threshold. However, making EE the primary basis for vulnerability prioritization is not advisable, as it considers vulnerabilities with less severe impacts on business operations, thereby increasing the effort required for remediation. In contrast, the proposed RISK R34P3R risk prioritization method combines the strengths of EPSS, EE, and CVSS. It considers the likelihood of exploitation (EPSS), ease of exploitation (EE), and severity impact ratings (CVSS). Fig. 5(c) illustrates that RISK R34P3R consistently strikes a balance, with 153 vulnerabilities falling within its remediation threshold. This approach reduces the additional effort needed to patch vulnerabilities when compared to relying on CVSS and EE alone. Moreover, it enhances coverage, addressing the limitations of EPSS. In summary, the findings emphasize that an integrated approach, such as RISK R34P3R, can offer a more efficient and comprehensive strategy for vulnerability prioritization, minimizing additional efforts while maintaining a high level of protection.

4.4. RQ4 - SQLi prevention

To address RQ4, we designed and implemented a prevention system that combines the detection, risk analysis and threat modelling components into a prevention mechanism. We discuss details of the design, implementation, and deployment of the mechanism, as well as the comparative analysis with state-of-the-art prevention systems below.

4.4.1. System design and implementation

The SQLR34P3R is a command line-enabled proxy solution designed for SQLi detection and prevention. A process diagram for deploying SQLR34P3R as a tool for the application layer SQLi attacks is illustrated in Fig. 6. The trained CNN-LSTM model is deployed within the proxy as an integral part of the detection and multiclass classification feature.

In Fig. 6, the regular user performs a legitimate login while the threat actor injects a malicious SQLi payload to bypass authentication. SQLR34P3R captures and examines HTTP requests from both the user and threat actor across multiple vectors, while also inspecting network traffic through inputted PCAP files for signs of DNS data exfiltration. If SQLR34P3R detects the request is benign, the request is then forwarded to the web server as normal, and an appropriate response is shown to the authorized user. However, if SQLR34P3R detects SQLi payload instances, the instances are categorized to their respective attack labels based on the characteristics specified in Table 3. The malicious HTTP request is prevented from reaching the backend web server and SQLR34P3R sends a response to the threat actor/web client notifying them such attacks are prohibited. Additionally, SQLR34P3R generates a risk rating using a novel vulnerability prioritization approach, RISK R34P3R, combining severity, exploitability, and exploit availability metrics. Threat Modelling is also performed by determining the source of the attack and by querying popular threat intelligence feeds to present the threat actor source, technique, and system information.

Fig. 7 presents a component diagram that illustrates the interactions among SQLR34P3R's features. It demonstrates how SQLR34P3R intercepts and processes the request through the various components in the event of an SQLi attack. The state diagram displayed in Fig. 8 illustrates SQLR34P3R's operational states. Upon execution, the proxy binds to port 8080 of the host system, however, if the port is in use, the execution is terminated. SQLR34P3R processes incoming web requests or inputted PCAP files for signs of SQLi attacks. Benign web traffic is forwarded to the web server as normal, while the detection of malicious web traffic ends the HTTP conversation and alerts the user. SQLR34P3R continues running until manually terminated by the system administrator. The CNN-LSTM model trained on network traffic data was deployed as an additional SQLR34P3R metric. Users can provide a PCAP capture file as input which will be parsed by the underlying Python code to detect any occurrence of SQLi DNS data exfiltration as shown by the model deployment diagram in Fig. 9.

System requirements

The system requirements for the proposed SQLR34P3R tool are as follows:

- SQLR34P3R must intercept and extract the HTTP URL, header, and body values from web requests.
- SQLR34P3R must load the CNN-LSTM model to classify SQLi payloads by attack type.
- SQLR34P3R must load the CNN-LSTM model to identify SQLi DNS exfiltration attacks using NetFlow data.
- SQLR34P3R must record the originating IP address of malicious requests.
- SQLR34P3R must generate risk rating using CVSS, EPSS and EE scores.
- SQLR34P3R must perform threat intelligence gathering on the originating IP address.
- SQLR34P3R must block the source IP address after three malicious attempts and drop the request.
- SQLR34P3R output correlated SQLi attack data in JSON format.

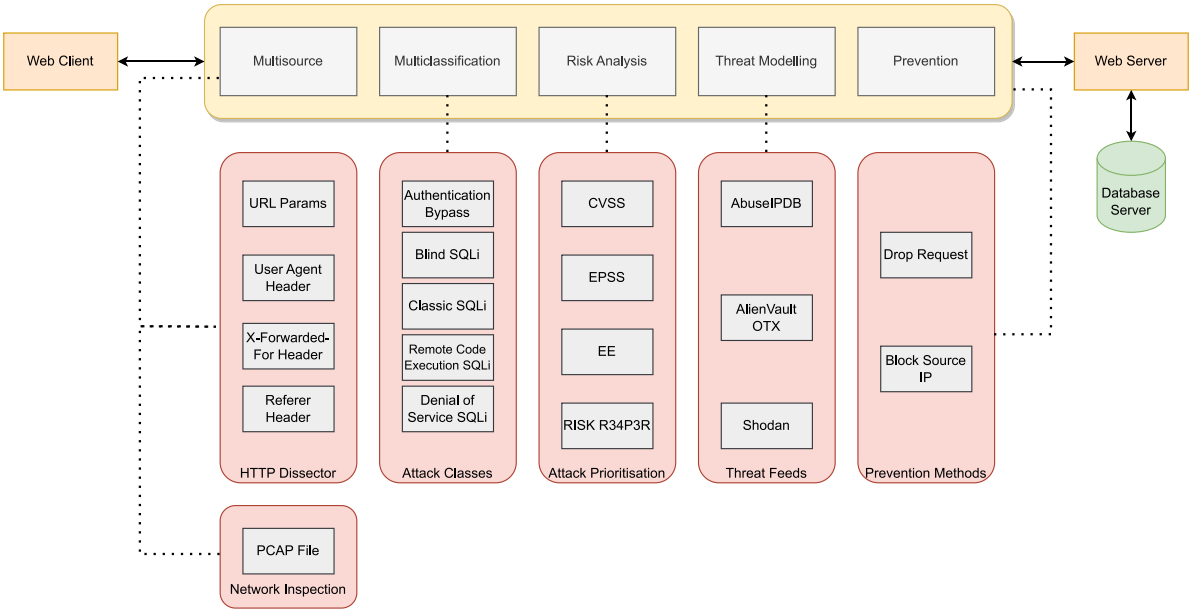


Fig. 7. SQLR34P3R: Component diagram.

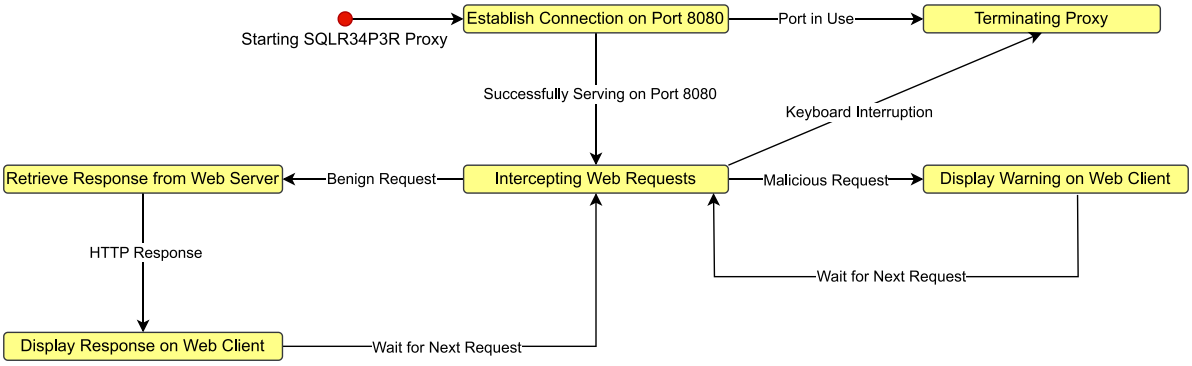


Fig. 8. SQLR34P3R: State diagram.

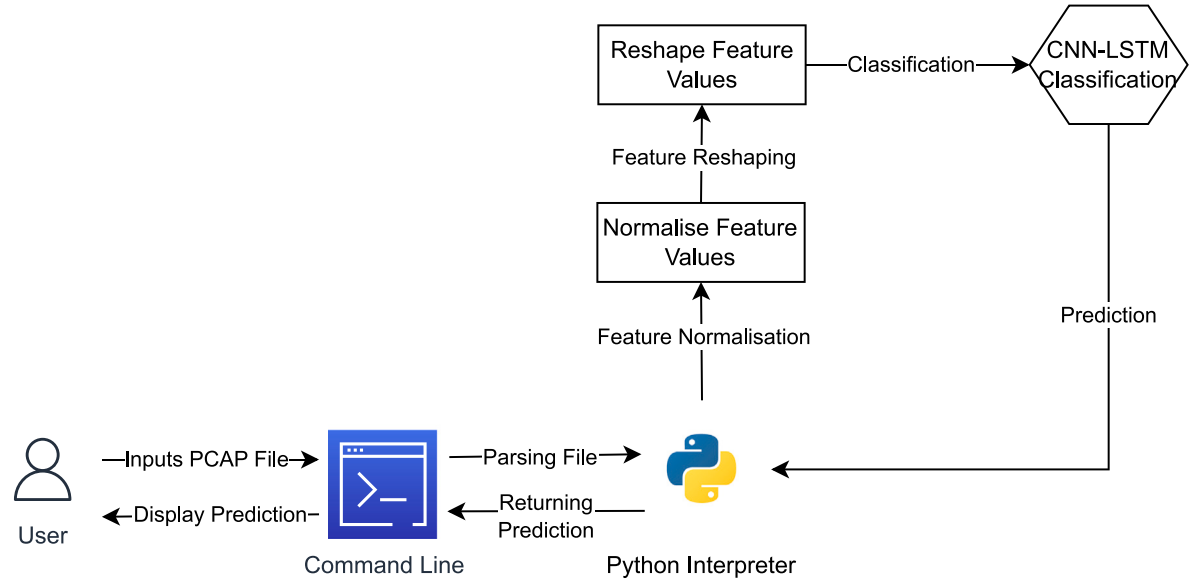


Fig. 9. SQLR34P3R: Model deployment.

Libraries used

SQLR34P3R was developed in Python 3.11.5, due to its extensive built-in library support which simplifies the implementation of machine and deep learning models. Python's readability, flexibility and data-handling capabilities demonstrate its superiority over alternative languages. Visual Studio Code (VS Code) was the chosen Integrated Development Environment (IDE) due to its portability and lightweight nature. Non-standard Python packages were installed using the package manager pip. The primary Python libraries that were used for the implementation are presented below:

- **Sklearn:** Sklearn or Scikit-learn is a comprehensive and community-driven machine learning library. Sklearn offers a plethora of machine learning functionality including SVM, and Random Forest to name a few [108].
- **Xgboost**: "xgboost" is a Python package which allows users to easily integrate the XGBoost algorithm into their program [109].
- **Requests**: Requests is a popular Python package used to handle HTTP-related tasks. It is possible to set custom headers, requests and body parameters using Requests [110].
- **Pandas**: Pandas is a data analysis and visualization library which makes it easy to work with structured data [108].
- **Mitmproxy**: Mitmproxy library is the Python integration for the Mitmproxy proxy which can intercept HTTP traffic and perform application layer packet analysis [111].
- **Tensorflow-Keras**: Tensorflow combines with Keras API to simplify the process of developing and training deep learning models [112].

Multisource enabled proxy

SQLR34P3R uses Python's subprocess module to launch "mitmdump" the command-line implementation of mitmproxy. Running the command "mitmdump –listen-host 0.0.0.0 -s proxy.py" using the subprocess module configures the proxy to listen on all network interfaces while executing the "proxy.py" Python script to capture the HTTP traffic between the web client and backend server. The method "http.HTTPFlow" intercepts and extracts HTTP header values from headers including "User-Agent", "Referer", and "X-Forwarded-For" in addition to URL query parameters and request body contents. Users can redirect traffic through the proxy by setting the browser to the SQLR34P3R host's IP address and port 8080. SQLR34P3R uses the Pyshark module to parse PCAP files and extract the network layer information such as the packet size, source port, destination port, and protocol type. The network layer information is then processed by the CNN-LSTM model to detect potential SQLi DNS data exfiltration.

Multiclass classification

Datasets collected from sources specified in Table 2 are aggregated and separated into six different text files influenced by literature and the unique characteristics established in Table 3. SQLR34P3R then loads the six files as individual DataFrames using the Pandas library and assigning labels between 0 and 5 to represent the six different classes. The six DataFrames are merged to form a single DataFrame that undergoes data pre-processing and feature extraction as described in Sections 3.1.3 and 3.1.4. For the traditional machine learning classifiers, both the models and the vectorizer are saved. In contrast, for the deep learning models, the models and the tokenizer are saved. During deployment, the optimal model (LSTM) is then loaded along with the tokenizer to perform classification tasks to detect and classify potential SQLi attacks.

Risk analysis implementation

SQLR34P3R's attack prioritization feature, RISK R34P3R, was developed by manually collecting CVEs for each class from the NIST database. Leveraging NIST's API, the CVSS scores for each CVE were extracted and converted to JSON file format using the "json_converter.py" script. The "risk_modelling.py" script uses the "calculate_cvss" function to calculate the average CVSS scores for each class. To determine the average EPSS score for each class, the "calculate_epss" function reads the CVE text files and queries the EPSS API to retrieve the individual EPSS scores. Similarly, for average EE score calculation, the EE API is invoked to extract scores for each SQLi class listed in the text files.

Threat intelligence integration

As detailed in Section 3.3, three threat intelligence feeds were integrated into SQLR34P3R to perform threat modelling: AbuseIPDB, Alienvault OTX, and Shodan. To incorporate AbuseIPDB, its provided API was implemented into the program. The AbuseIPDB API facilitated the retrieval of various details related to the SQLi attack origin and source IP including the originating country, hostname, associated domains, TOR associations and malicious reports. Alienvault OTX were used to profile the threat actor by invoking their individual APIs. Finally, to gain insight into the threat actor's system, the Shodan API was utilized to identify open ports and vulnerabilities.

Prevention system

To address RQ4, the implemented active response and prevention feature will immediately return a 403 Forbidden status code and display the message "Your IP has been blocked due to suspicious of SQLi Attack. Request dropped" using the "http.Response.make" function after three SQLi attempts. This ensures the malicious request does not reach the backend database by automatically dropping the request.

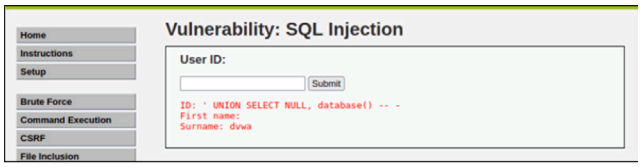
4.4.2. System demonstration

The lab environment setup and configuration for the system demonstration, which includes the software and network configuration is described as follows.

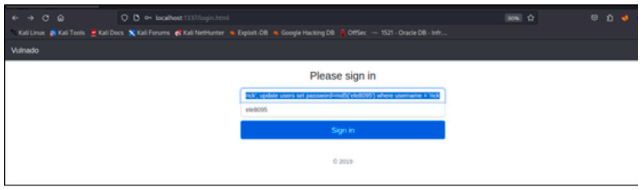
- **Kali Linux (2022.3)**: Kali Linux [113] is the host running the SQLR34P3R proxy.
- **Ubuntu (12.04)**: The SQLi attacks will be initiated from the Ubuntu system [114] posing as the threat actor.
- **Metasploitable 2**: This is a purposefully insecure Ubuntu Linux virtual machine designed for testing security tools, demonstrating common vulnerabilities, and practising common penetration testing techniques [115].
- **Damn Vulnerable Web Application**: DVWA [116] is an intentionally vulnerable PHP application from practising common web vulnerabilities. DVWA is hosted on Metasploitable 2.
- **Vulnado**: Vulnado [117] is an intentionally vulnerable Java application hosted on Docker.

To demonstrate the threat modelling feature, a public IP was manually assigned to the threat actor Ubuntu machine, given the constraints of public IP assignments by Internet Service Providers (ISPs) in internal lab settings.

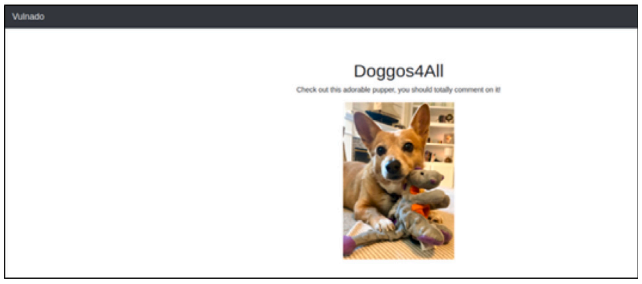
Fig. 10 demonstrated the SQLi vulnerabilities in the purposefully vulnerable web and Java applications. The presence of the SQLi vulnerability within DVWA can be initially confirmed by performing an SQLi attack on the "User ID" input using the payload 'UNION SELECT NULL, database() – -. Fig. 10(a) confirms that DVWA is susceptible to SQLi attacks. Fig. 10(b) shows that the "/login.html" endpoint within the Java application Vulnado is vulnerable to authentication bypass vulnerability by executing the payload "rick'; update users set password=md5('ele8095') where username = 'rick' –" which will change the password for the user rick to "ele8095". After executing the payload shown in Fig. 10(b), the threat actor can bypass authentication by entering the updated password for the rick user and successfully logging in to the application as shown in Fig. 10(c).



(a) DVWA SQLi Vulnerability

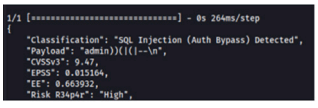


(b) Vulnado Vulnerable Endpoint



(c) Authentication Bypassed

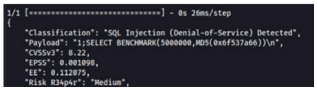
Fig. 10. Demonstrating DVWA & Vulnado SQLi vulnerabilities.



(a) Authentication Bypass SQLi Detection and Classification

(b) Remote Code Execution SQLi Detection and Classification

(c) Blind SQLi Detection and Classification



(d) Denial-of-Service SQLi Detection and Classification

Fig. 11. Detection and classification of various SQLi attacks.

SQLi attack detection, multiclass classification and risk analysis (HTTP flow)

To demonstrate the functionality of SQLR34P3R, traffic from two separate web applications (DVWA & Vulnado) will be proxied through SQLR34P3R. It will inspect the HTTP flow for SQLi attack patterns using the CNN-LSTM model, showcasing the solution's language-agnostic nature. Payloads were selected that are not present in the datasets to demonstrate the detection and classification of unseen SQLi attacks into their classes. In Fig. 11, the successful detection and classification of various SQLi attacks using specific payloads are demonstrated as follows.

In Figs. 11(a), 11(b), 11(c) and 11(d), the unique CVSS, EPSS, EE, and RISK R34P3R scores for each attack class are also showcased, highlighting the severity of each detected SQLi attack class. Fig. 12 presents the threat modelling metrics for a detected SQLi attack. This figure features various reports: the "IP Report" (detailing the source of the attack), the "Threat Report" (providing a threat actor profile), and the "System Report" (offering information on the threat actor's system). Finally, the complete SQLR34P3R output combines the detection/classification, risk analysis, and threat modelling results.

Fig. 12. Threat modelling output for detected SQLi attack.

Multisource (DNS exfiltration and header injection)

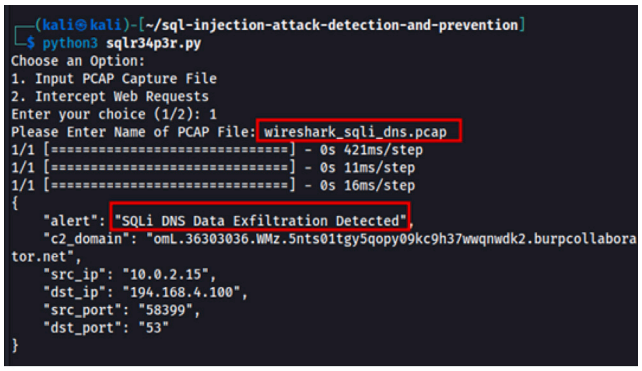
To demonstrate DNS data exfiltration during SQLi attacks via NetFlow data, network traffic was captured using Wireshark [118] from SQLi DNS exfiltration conducted with SQLMap. Standard DNS traffic was also captured. Both datasets were saved as PCAP files for analysis. The PCAP file containing SQLi DNS exfiltration network traffic covers a time window of one minute and nine seconds, while the PCAP file with standard DNS traffic spans a 13-second time frame. Fig. 13 shows the results of the system evaluation for SQLi detection in Network traffic. Fig. 13(a) displays SQLR34P3R's output upon detecting SQLi DNS Exfiltration, highlighting the destination domain, IP address, and both source and destination ports used in the transfer. After inputting the benign capture file, SQLR34P3R successfully outputs "No SQLi DNS Data Exfiltration Detected" as shown in Fig. 13(b). Additionally, SQLR34P3R detects SQLi attacks through vectors like User-Agent, Referer and X-Forwarded-For headers as shown in Fig. 14. Figs. 14(a), 14(b) and 14(c) demonstrate the blind SQLi-based header injection. To differentiate between each header injection, each payload within the headers has been assigned a unique time element in the "waitfor" payload as shown in Fig. 15. Figs. 15(a), 15(b) and 15(c) demonstrate the successful detection and multiclass classification of the SQLi attacks.

Prevention (request filtering through IP blocking)

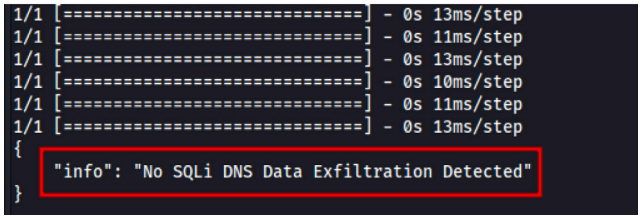
Fig. 16 displays the successful blocking of the IP address after three malicious SQLi attack attempts. The request is dropped and prevented from reaching the backend server. A warning in JSON format is displayed to the potential threat actor.

4.4.3. System comparative analysis

The proposed solution, SQLR34P3R, was evaluated against the features of the state-of-the-art SQLi attack detection and prevention solutions in Table 11. SQLR34P3R presents a holistic approach when compared with the current state-of-the-art solutions listed. Most of the frameworks including synBERT and PNN are trained only on malicious and benign HTTP traffic and can only perform binary classification. In contrast, SQLR34P3R can detect application and network layer SQLi attacks and can attribute different SQLi attacks to their respective types. None of the evaluated solutions implement risk analysis and threat modelling features while SQLR34P3R performs risk analysis using a novel approach, RISK R34P3R which combines the advantages of CVSS, EPSS and EE while performing threat profiling of the source of attack. It was observed that only 30% of the state-of-the-art solutions implemented proactive prevention methods. However, SQLR34P3R implements active response capabilities, blocking IP addresses after consecutive malicious attempts and discarding the request before it reaches the backend.

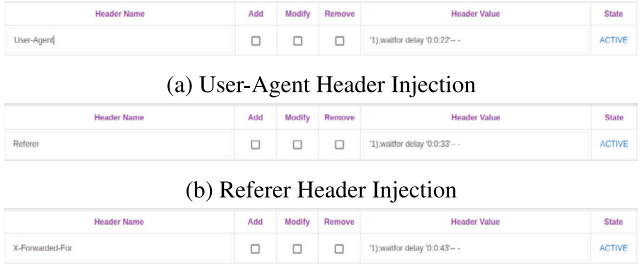


(a) SQLi DNS Data Exfiltration Detection



(b) No SQLi Exfiltration Detected

Fig. 13. Demonstrating SQLi detection in network traffic.

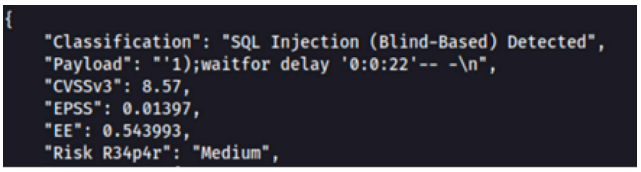


(a) User-Agent Header Injection

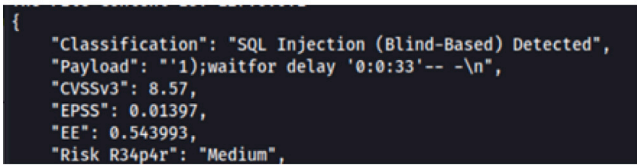
(b) Referer Header Injection

(c) X-Forwarded-For Header Injection

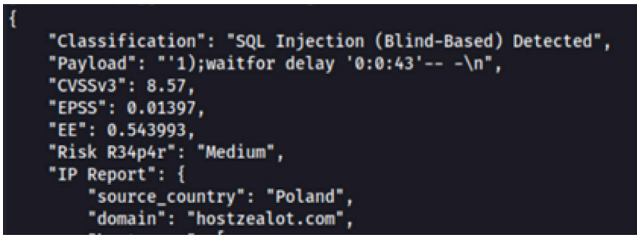
Fig. 14. System evaluation blind SQLi-based header injection.



(a) SQLR34P3R: User-Agent SQLi Detection



(b) SQLR34P3R Referer SQLi Detection



(c) SQLR34P3R: X-Forwarded-For SQLi Detection

Fig. 15. System evaluation header injection.

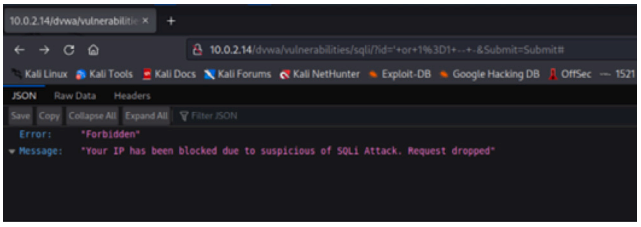


Fig. 16. SQLR34P3R prevention through IP blocking and request filtering.

5. Challenges and limitations

In this section, we discuss the limitations of SQLR34P3R and possible future work in this area. Currently, SQLR34P3R analyzes only HTTP requests for SQL injection (SQLi) attack detection, which is a common practice [21]. However, this study did not specifically address other potential data sources such as IP addresses, other transaction signals, HTTP responses, or the detection of queries being smuggled at the database wire protocol level. The unavailability of multi-class NetFlow datasets has prevented the implementation of multi-class detection for network traffic-based SQLi attack detection. Furthermore, the models for both SQLi payload and network traffic were deployed individually due to differing dataset structures, which has hindered the development of a correlated model.

In the multiclass classification of SQLi attacks, one challenge is the difference in dataset sizes. The samples collected for SQLi authentication bypass, remote code execution, and denial of service were smaller compared to the other categories, as shown in Table 2. This is because current approaches treat the SQLi detection problem as a binary classification issue, gathering benign and malicious queries without considering the sophistication of the attack vector [122]. Thus, no benchmark datasets were available for our approach, prompting the creation of a multiclass dataset for this research. This necessitates broader data collection from diversified and trusted sources. We have used the F1-Score, the most common metric on imbalanced classification problems. A worthy research direction is investigating how the class size affects the F1-Score. It is commonly misunderstood that the performance of an ML model is directly proportional to the size of the training data [123,124]. However, we can use experimental validation from the deployed model as a preventive measure for SQLi attack (see Fig. 9) to determine the optimal training set size. In the current research, there is no feedback loop between the deployed model and the training set.

One limitation of implementing IP blocking as a prevention method is its potential to adversely affect multiple users within large organizations. For example, if an individual from an organization performs an SQL injection attack on an external system, the organization's public IP address may be blocked. This would unintentionally prevent all members of that organization from accessing the external system, as the defence mechanism only identifies the shared public IP address of the organization. In a university setting, if a student from University X is conducting an SQL injection attack against System Y hosted by Company Z, the student's public IP address would be blocked. However, this action would prevent all students from University X from accessing the system, as the defence system would only recognize the public IP address of the university.

The proposed SQLi vulnerability prioritization approach utilizes scoring systems like CVSS, EPSS, and EE as data points for decision-making. These scoring systems are part of the key risk factor standards for risk-based vulnerability management, which is based on exploit

Table 11
State-of-the-Art (SOTA) comparative analysis with SQLR34P3R.

SOTA	Detection method	Data source	Risk analysis	Threat modelling	Prevention
[22]	Binary	HTTP	✗	✗	✓
[23]	Binary	HTTP	✗	✗	✓
[24]	Binary	HTTP	✗	✗	✓
[25]	Binary	HTTP	✗	✗	✗
[27]	Binary	HTTP	✗	✗	✗
[28]	Binary	HTTP	✗	✗	✗
[29]	Binary	HTTP	✗	✗	✗
[30]	Binary	HTTP, Network Traffic	✗	✗	✗
[31]	Binary	HTTP Only	✗	✗	✓
[32]	Binary	HTTP Only	✗	✗	✓
[34]	Binary	HTTP Only	✗	✗	✓
[35]	Binary	Network Traffic	✗	✗	✗
[36]	Binary	HTTP, Network Traffic	✗	✗	✗
[37]	Binary	HTTP	✗	✗	✗
[26]	Binary	HTTP	✗	✗	✗
[38]	Binary	HTTP	✗	✗	✗
[119]	Binary	HTTP, Network Traffic	✓	✗	✗
[120]	Binary	HTTP	✗	✗	✗
[121]	Binary	HTTP	✗	✗	✗
[33]	Binary	Network Traffic	✗	✗	✗
Ours	Binary, Multiclass	HTTP, Network Traffic	✓	✓	✓

evidence [125,126]. The current system for prioritizing vulnerabilities does not take into account the specific environment and assets when assessing SQL vulnerabilities. Additionally, it does not consider exploit intelligence when evaluating the threat from SQL injection (SQLi) attacks. To address this, future development will focus on improving the vulnerability prioritization system by incorporating the Known Exploited Vulnerabilities Catalog (KEV), such as *CISA* and *VulnCheck* KEV catalog. This will provide visibility into SQLi vulnerabilities that are currently being exploited by Ransomware, Botnets, threat actors, and unclassified sources [125]. This integration will allow for prioritizing CVEs found in the catalog, improving the effectiveness of the system. In addition to prioritizing known exploited SQLi vulnerabilities, we will also consider weaponized SQLi vulnerabilities, the availability of exploit code for SQLi vulnerabilities, and the specific environment and assets involved. This will provide a more accurate assessment of the true risk posed by an SQLi vulnerability within a given environment [125].

Additionally, the research's approach to managing SQL injection (SQLi) vulnerabilities is focused on Common Vulnerabilities and Exposures (CVE) data. This means it relies on NIST National Vulnerability Database (NVD) APIs to gather information about SQLi vulnerabilities. However, this approach can lead to delays in analysing vulnerabilities and reliability issues stemming from limitations in the NVD's coverage and visibility of CVEs related to SQLi. A case in point is the NVD's February 2024 announcement of service degradation, which led to delays in processing CVE data and important metadata such as details about affected software, CVSS scores, and severity. This resulted in a gap in the public vulnerability dataset [127,128]. Finally, SQLR34P3R does not currently perform real-time network traffic inspection. It only takes previously captured PCAP files as input. In future implementations, real-time network traffic inspection will be added to enhance detection capabilities.

6. Conclusion

This research proposes a comprehensive approach to detecting SQL injection attacks. It begins with conducting an analysis of multiple data sources and evaluating traditional machine learning, deep learning, and hybrid learning techniques for this purpose. Additionally, a novel risk analysis approach is introduced, which involves combining scoring systems such as CVSS, EPSS, and EE scores with threat intelligence to prioritize remediation efforts for SQL injection vulnerability patching efficiently. Furthermore, the functionality for detection, classification, and prioritization is enhanced with preventive response measures. The research also introduces SQLR34P3R, a command-line proxy that plays a crucial role in SQL injection detection and prevention. It offers real-time web filtering and PCAP analysis capabilities to prevent SQL injection data exfiltration through DNS. The three-pronged approach of multisource detection, multiclass classification, and comprehensive risk assessment sets SQLR34P3R apart from existing state-of-the-art solutions, demonstrating its potential as a valuable defence against SQL injection attacks.

The following areas of research could be explored further based on this work. It would be valuable to investigate an evidence-based approach to prioritizing SQL injection (SQLi) vulnerabilities, using exploit evidence and threat context. Additionally, it would be beneficial to explore preventive measures for SQL injection attacks (SQLiA) beyond the current reliance on IP blocking, which may be inadequate for large organizations. For example, using parameterized query fingerprinting as a defence mechanism to detect SQL injection in log sources and telemetry can improve current defence techniques. Future work could focus on real-time detection of network traffic, as opposed to the current near-real-time approach of analysing captured packets (PCAP) for improved effectiveness. To effectively prioritize vulnerabilities, consideration of comprehensive contextual factors, such as environmental, asset, runtime, application, and business contexts, is crucial. Decision-based frameworks like Stakeholder-Specific Vulnerability Categorization should be used to incorporate these contexts and assist security practitioners in prioritizing vulnerabilities.

CRediT authorship contribution statement

Alan Paul: Writing – original draft, Software, Methodology, Investigation, Data curation, Conceptualization. **Vishal Sharma:** Writing – review & editing, Validation, Project administration, Methodology, Investigation, Conceptualization. **Oluwafemi Olukoya:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Conceptualization.

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.

Data availability

No data was used for the research described in the article.

References

[1] Nivedita J. 160 Cybersecurity statistics 2023 [updated]. 2023, Astra IT, https://www.getastra.com/blog/security-audit/cyber-security-statistics/.
[2] OWASP. OWASP top ten. 2023, The OWASP® Foundation, https://owasp.org/www-project-top-ten/.
[3] Ilascu I. WooCommerce fixes vulnerability exposing 5 million sites to data theft. 2021, Bleeping Computer, https://www.bleepingcomputer.com/news/security/woocommerce-fixes-vulnerability-exposing-5-million-sites-to-data-theft/.
[4] Cimpanu C. Hackers use SQL injection bug in BillQuick billing app to deploy ransomware. 2021, Recorded Future News, https://therecord.media/hackers-use-sql-injection-bug-in-billquick-billing-app-to-deploy-ransomware.
[5] Sharma A. Django fixes SQL injection vulnerability in new releases. 2022, Bleeping Computer, https://www.bleepingcomputer.com/news/security/django-fixes-sql-injection-vulnerability-in-new-releases/.
[6] Hanley Z. MOVEit Transfer CVE-2023-34362 deep dive and indicators of compromise. 2023, horizon3.ai, https://www.horizon3.ai/moveit-transfer-cve-2023-34362-deep-dive-and-indicators-of-compromise/.

[7] Fortra Security. SQL Injection Vulnerability in FileCatalyst Workflow 5.1.6 Build 135 (and earlier). 2024, Fortra, https://www.fortra.com/security/advisory/fi-2024-008.

[8] Horseman J. CVE-2024-29824 deep dive: Ivanti EPM SQL injection remote code execution vulnerability. 2024, horizon3.ai, https://www.horizon3.ai/attack-research/attack-blogs/cve-2024-29824-deep-dive-ivanti-epm-sql-injection-remote-code-execution-vulnerability/.

[9] Zhu Z, Jia S, Li J, Qin S, Guo H. SQL injection attack detection framework based on HTTP traffic. In: Proceedings of the ACM turing award celebration conference-China. 2021, p. 179–85.

[10] Irungu J, Graham S, Girma A, Kacem T. Artificial intelligence techniques for SQL injection attack detection. In: Proceedings of the 2023 8th international conference on intelligent information technology. 2023, p. 38–45.

[11] Alwan ZS, Younis MF. Detection and prevention of SQL injection attack: a survey. Int J Comput Sci Mobile Comput 2017;6(8):5–17.

[12] Singh JP. Analysis of SQL injection detection techniques. 2016, arXiv preprint arXiv:1605.02796.

[13] Stampar M. Data retrieval over DNS in SQL injection attacks. 2013, arXiv preprint arXiv:1303.3047.

[14] MITRE. 2023 CWE top 25 most dangerous software weaknesses. 2023, https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html, Accessed: 2023-10-27.

[15] MITRE. Stubborn weaknesses in the CWE top 25. 2023, https://cwe.mitre.org/top25/archive/2023/2023_stubborn_weaknesses.html, Accessed: 2023-10-27.

[16] CISA. Secure by design alert: Eliminating SQL injection vulnerabilities in software. 2024, CISA Central, https://www.cisa.gov/resources-tools/resources/secure-design-alert-eliminating-sql-injection-vulnerabilities-software.

[17] Oftendal E, Baars N. SQL injection isn't dead yet. 2024, OWASP Foundation, https://dev.to/owasp/sql-injection-isnt-dead-yet-48ic.

[18] IppSec. Automating boolean SQL injection and evading filters. 2023, https://www.youtube.com/watch?v=mF8Q1FhnU70&t=3290s, Digital Threats Accessed: 2023-10-26.

[19] Google-Bug-Hunters. Verify the output of the tools. 2023, https://bughunters.google.com/learn/improving-your-reports/avoiding-mistakes/5981856648134656/verify-the-output-of-the-tools, Accessed: 2023-10-26.

[20] Perkal Y. Scanning the scanners: What vulnerability scanners miss and why — And what this means for your software attack surface. 2022, https://info.rezilion.com/scanning-the-scanners-what-vulnerability-scanners-miss-and-why/, Accessed: 2023-10-26.

[21] Moissinac B, Saad E, Clay M, Berrondo M. Detecting SQL injection attacks using machine learning. In: CAMLIS. 2023, p. 49–59.

[22] Halfond WG, Orso A. Preventing SQL injection attacks using AMNESIA. In: Proceedings of the 28th international conference on software engineering. 2006, p. 795–8.

[23] Hadabi A, Elsamani E, Abdallah A, Elhabob R. An efficient model to detect and prevent SQL injection attack. J Karary Univ Eng Sci 2022.

[24] Kini S, Patil AP, Pooja M, Balasubramanyam A. SQL injection detection and prevention using Aho-Corasick pattern matching algorithm. In: 2022 3rd international conference for emerging technology. INCET, IEEE; 2022, p. 1–6.

[25] Fu X, Qian K. SAFELI: SQL injection scanner using symbolic execution. In: Proceedings of the 2008 workshop on testing, analysis, and verification of web services and applications. 2008, p. 34–9.

[26] Lu D, Fei J, Liu L. A semantic learning-based SQL injection attack detection technology. Electronics 2023;12(6):1344.

[27] Siddiq ML, Jahin MRR, Islam MRU, Shahriyar R, Iqbal A. Sqlifix: Learning based approach to fix sql injection vulnerabilities in source code. In: 2021 IEEE international conference on software analysis, evolution and reengineering. SANER, IEEE; 2021, p. 354–64.

[28] Kar D, Sahoo AK, Agarwal K, Panigrahi S, Das M. Learning to detect SQLIA using node centrality with feature selection. In: 2016 international conference on computing, analytics and security trends. CAST, IEEE; 2016, p. 18–23.

[29] Ladole A, Phalke M. SQL injection attack and user behavior detection by using query tree, fisher score and SVM classification. Int Res J Eng Technol 2016;3(6):1505–9.

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

[31] Uwagbole SO, Buchanan WJ, Fan L. 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; 2017, p. 1087–90.

[32] Jahanshahi R, Doupé A, Egele M. You shall not pass: Mitigating sql injection attacks on legacy web applications. In: Proceedings of the 15th ACM Asia conference on computer and communications security. 2020, p. 445–57.

[33] Crespo-Martínez IS, Campazas-Vega A, Guerrero-Higueras ÁM, Riego-DelCastillo V, Álvarez-Aparicio C, Fernández-Llamas C. SQL injection attack detection in network flow data. Comput Secur 2023;127:103093.

[34] Chen D, Yan Q, Wu C, Zhao J. Sql injection attack detection and prevention techniques using deep learning. In: Journal of physics: conference series. 1757, (1):IOP Publishing; 2021, 012055.

[35] Luo A, Huang W, Fan W. A CNN-based approach to the detection of SQL injection attacks. In: 2019 IEEE/aCIS 18th international conference on computer and information science. ICIS, IEEE; 2019, p. 320–4.

[36] Li Q, Wang F, Wang J, Li W. LSTM-based SQL injection detection method for intelligent transportation system. IEEE Trans Veh Technol 2019;68(5):4182–91.

[37] Tang P, Qiu W, Huang Z, Lian H, Liu G. Detection of SQL injection based on artificial neural network. Knowl-Based Syst 2020;190:105528.

[38] Alarfaj FK, Khan NA. Enhancing the performance of SQL injection attack detection through probabilistic neural networks. Appl Sci 2023;13(7):4365.

[39] Nasereddin M, ALKhamaiseh A, Qasaimeh M, Al-Qassas R. A systematic review of detection and prevention techniques of SQL injection attacks. Inform Secur J: A Global Perspective 2023;32(4):252–65.

[40] Alghawazi M, Alghazzawi D, Alarifi S. Detection of SQL injection attack using machine learning techniques: a systematic literature review. J Cybersecur Priv 2022;2(4):764–77.

[41] Qbea'h M, Alrabaee S, Alshraideh M, Sabri KE. Diverse approaches have been presented to mitigate SQL injection attack, but it is still alive: A review. In: 2022 international conference on computer and applications. ICCA, IEEE; 2022, p. 1–5.

[42] Damele A. G. B, Stampar M. Sqlmap: Automatic SQL injection and database takeover tool. 2023, sqlmapproject, https://github.com/sqlmapproject/sqlmap.

[43] Ojagbule O, Wimmer H, Haddad RJ. Vulnerability analysis of content management systems to SQL injection using SQLMAP. In: SoutheastCon 2018. IEEE; 2018, p. 1–7.

[44] Campazas-Vega A, Crespo-Martínez IS, Manuel G-HÁ. Docker-based framework for gathering netflow data (DOROTHEA). Zenodo; 2020, http://dx.doi.org/10.5281/zenodo.4114119.

[45] Crespo I, Campazas A. SQL injection attack netflow. 2022, zenodo, https://zenodo.org/record/6907252.

[46] Shala A. SQL injection authentication bypass payloads. 2020, GitHub, Inc., https://gist.github.com/spenkk/2cd2f7eeb9cac92dd550855e522c558f.

[47] Taşdelen I. SQL injection payloads list. 2021, GitHub, Inc., https://github.com/payloadbox/sql-injection-payload-list/blob/master/Intruder/exploit/Auth_Bypass.txt.

[48] Polop C. SQL login bypass. 2023, GitHub, Inc., https://github.com/carlospolop/hacktricks/blob/master/pentesting-web/login-bypass/sql-login-bypass.md.

[49] OWASP. Testing for SQL server. The OWASP® Foundation; 2023, https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05.3-Testing_for_SQL_Server.

[50] Shah SSH. Sql injection dataset. 2021, kaggle, https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset.

[51] Nagpal B, Chauhan N, Singh N. A survey on the detection of SQL injection attacks and their countermeasures. J Inform Proc Syst 2017;13(4).

[52] Moldovan F, Sătmărean P, Oprişa C. An analysis of http attacks on home iot devices. In: 2020 IEEE international conference on automation, quality and testing, robotics. AQTR, IEEE; 2020, p. 1–6.

[53] Wang M, Jung C, Ahad A, Kwon Y. Spinner: Automated dynamic command subsystem perturbation. In: Proceedings of the 2021 ACM SIGSAC conference on computer and communications security. 2021, p. 1839–60.

[54] Campazas-Vega A, Crespo-Martínez IS, Guerrero-Higueras ÁM, Álvarez-Aparicio C, Matellán V. Analysis of netflow features' importance in malicious network traffic detection. In: 14th international conference on computational intelligence in security for information systems and 12th international conference on European transnational educational (CISIS 2021 and ICEUTE 2021) 14. Springer; 2022, p. 52–61.

[55] Shareef OSF, Hasan RF, Farhan AH. Analyzing SQL payloads using logistic regression in a big data environment. J Intell Syst 2023;32(1):20230063.

[56] Ghozali I, Asy'ari MF, Triarjo S, Ramadhani HM, Studiawan H, Shiddiqi AM. A novel SQL injection detection using Bi-LSTM and TF-IDF. In: 2022 7th international conference on information and network technologies. ICINT, IEEE; 2022, p. 16–22.

[57] Dharma EM, Gaol FL, Warnars H, Soewito B. The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification. J Theor Appl Inf Technol 2022;100(2):31.

[58] Pennington J, Socher R, Manning CD. Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing. EMNLP, 2014, p. 1532–43.

[59] Ferreira P, Le DC, Zincir-Heywood N. Exploring feature normalization and temporal information for machine learning based insider threat detection. In: 2019 15th international conference on network and service management. CNSM, IEEE; 2019, p. 1–7.

[60] Surles W. Machine learning toolbox. RpUBS; 2017, https://rpubs.com/williamsurles/310197.

[61] Almourish MH, Abduljalil OA, Alawi AEB. Anomaly-based web attacks detection using machine learning. In: International conference on smart computing and cyber security: strategic foresight, security challenges and innovation. Springer; 2021, p. 306–14.

[62] Demilie WB, Deriba FG. Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques. J Big Data 2022;9(1):124.

[63] Zhang W, Li Y, Li X, Shao M, Mi Y, Zhang H, Zhi G, et al. Deep neural network-based SQL injection detection method. Secur Commun Netw 2022;2022.

[64] Gholamy A, Kreinovich V, Kosheleva O. Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation. In: Departmental technical reports. CS, 1209, 2018.

[65] Nguyen QH, Ly H-B, Ho LS, Al-Ansari N, Le HV, Tran VQ, Prakash I, Pham BT. Influence of data splitting on performance of machine learning models in prediction of shear strength of soil. Math Probl Eng 2021;2021:1–15.

[66] Shhadat I, Hayajneh A, Al-Sharif ZA, et al. The use of machine learning techniques to advance the detection and classification of unknown malware. Procedia Comput Sci 2020;170:917–22.

[67] Ismail AA, Yusoff M. An efficient hybrid LSTM-CNN and CNN-LSTM with glove for text multi-class sentiment classification in gender violence. Int J Adv Comput Sci Appl 2022;13(9).

[68] Gandhi N, Patel J, Sisodiya R, Doshi N, Mishra S. A CNN-BiLSTM based approach for detection of SQL injection attacks. In: 2021 international conference on computational intelligence and knowledge economy. ICCIKE, IEEE; 2021, p. 378–83.

[69] Kim T-Y, Cho S-B. Optimizing CNN-LSTM neural networks with PSO for anomalous query access control. Neurocomputing 2021;456:666–77.

[70] Petmezas G, Haris K, Stefanopoulos L, Kilintzis V, Tzavelis A, Rogers JA, Katsaggelos AK, Maglaveras N. Automated atrial fibrillation detection using a hybrid CNN-LSTM network on imbalanced ECG datasets. Biomed Signal Process Control 2021;63:102194.

[71] Rehman AU, Malik AK, Raza B, Ali W. A hybrid CNN-LSTM model for improving accuracy of movie reviews sentiment analysis. Multimedia Tools Appl 2019;78:26597–613.

[72] Alhussein M, Aurangzeb K, Haider SI. Hybrid CNN-LSTM model for short-term individual household load forecasting. IEEE Access 2020;8:180544–57.

[73] She X, Zhang D. Text classification based on hybrid CNN-LSTM hybrid model. In: 2018 11th international symposium on computational intelligence and design. ISCID, 2, IEEE; 2018, p. 185–9.

[74] Mohajon J. Confusion matrix for your multi-class machine learning model. 2020, https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826.

[75] Torkura KA, Sukmana MI, Kayem AV, Cheng F, Meinel C. A cyber risk based moving target defense mechanism for microservice architectures. In: 2018 IEEE intl conf on parallel & distributed processing with applications, ubiquitous computing & communications, big data & cloud computing, social computing & networking, sustainable computing & communications (ISPA/iUCC/bDCloud/socialCom/sustainCom). IEEE; 2018, p. 932–9.

[76] Mell P, Scarfone K, Romanosky S. Common vulnerability scoring system. IEEE Secur Priv 2006;4(6):85–9.

[77] Spring J, Hatleback E, Householder A, Manion A, Shick D. Time to change the CVSS? IEEE Security & Privacy 2021;19(2):74–8.

[78] Tai W. What is VPR and how is it different from CVSS?. Tenable; 2020, https://www.tenable.com/blog/what-is-vpr-and-how-is-it-different-from-cvss.

[79] Howland H. Cvss: Ubiquitous and broken. Digit Threats: Res Practice 2023;4(1):1–12.

[80] Jacobs J, Romanosky S, Edwards B, Adjerid I, Roytman M. Exploit prediction scoring system (epss). Digit Threat: Res Pract 2021;2(3):1–17.

[81] Jacobs J, Romanosky S, Suciu O, Edwards B, Sarabi A. Enhancing vulnerability prioritization: Data-driven exploit predictions with community-driven insights. In: 2023 IEEE European symposium on security and privacy workshops (euroS&pW). IEEE; 2023, p. 194–206.

[82] Suciu O, Nelson C, Lyu Z, Bao T, Dumitraş T.

[83] Williams J. OWASP risk rating methodology. 2023, The OWASP® Foundation, https://owasp.org/www-community/OWASP_Risk_Rating_Methodology.

[84] Yermalovich P, Mejri M. Risk forecasting automation on the basis of MEHARI. In: Information and cyber security: 19th international conference, iSSA 2020, pretoria, South africa, August 25–26, 2020, revised selected papers 19. Springer; 2020, p. 34–49.

[85] Jung B, Li Y, Bechor T. CAVP: A context-aware vulnerability prioritization model. Comput Secur 2022;116:102639.

[86] Suciu O. Expected exploitability. Maryland Cybersecurity Center; 2023, https://www.exploitability.app/.

[87] Le TH, Chen H, Babar MA. A survey on data-driven software vulnerability assessment and prioritization. ACM Comput Surv 2022;55(5):1–39.

[88] Cipollone F. Why prioritize vulnerability? A case for risk and contextual-based prioritization for application security and cloud security. 2020, Phoenix Security, https://appsecphoenix.com/prioritize-vulnerabilities-risk-context-vulnerability-appsec-cloudsec/.

[89] Bobbitt Z. How to normalize data between 0 and 100. 2020, Maryland Cybersecurity Center; https://www.statology.org/normalize-data-between-0-and-100/.

[90] Chinchilla MR. CVE prioritizer tool. 2023, GitHub, https://github.com/TURROKS/CVE_Prioritizer.

[91] CISAgov. Known exploited vulnerabilities catalog. 2023, CISA Central, https://www.cisa.gov/known-exploited-vulnerabilities-catalog.

[92] Evans J, Baker J, Struse R. CVE + MITRE ATT&CK to understand vulnerability impact. 2021, MITRE-Engenuity, https://medium.com/mitre-engenuity/cve-mitre-att-ck-to-understand-vulnerability-impact-c40165111bf7.

[93] MITREorg. Mapping ATT&CK to CVE for impact. MITRE-Engenuity; 2021, https://mitre-engenuity.org/cybersecurity/center-for-threat-informed-defense/our-work/mapping-attck-to-cve-for-impact/.

[94] Abdeen B, Al-Shaer E, Singhal A, Khan L, Hamlen K. SMET: Semantic mapping of CVE to ATT&CK and its application to cybersecurity. In: IFIP annual conference on data and applications security and privacy. Springer; 2023, p. 243–60.

[95] Kuppa A, Aouad L, Le-Khac N-A. Linking cve's to mitre att&ck techniques. In: Proceedings of the 16th international conference on availability, reliability and security. 2021, p. 1–12.

[96] Grigorescu O, Nica A, Dascalu M, Rughinis R. Cve2att&ck: Bert-based mapping of cves to mitre att&ck techniques. Algorithms 2022;15(9):314.

[97] Marathon-Studios-Inc. AbuseIPDB: making the internet safer, one IP at a time. 2023, AbuseIPDB LLC, https://www.abuseipdb.com/.

[98] Ando R, Itoh H. Characterizing combatants of state-sponsored APT in digital warfare by reported blocklist database. IJCSNS 2022;22(3):541.

[99] AT&T-Cybersecurity. AlienVault Open Threat Exchange (OTX). 2023, ALIENVAULT, INC. https://otx.alienvault.com/.

[100] Lutf M. Threat intelligence sharing: a survey. J Appl Sci Comput 2018;8(11):1811–5.

[101] Matherly J. Shodan: Search engine for the internet of everything. 2023, Shodan, https://www.shodan.io/.

[102] Bada M, Pete I. An exploration of the cybercrime ecosystem around Shodan. In: 2020 7th international conference on internet of things: systems, management and security. IOTSMS, IEEE; 2020, p. 1–8.

[103] Liu J, Shen Y, Yan H. Functions-based cfg embedding for malware homology analysis. In: 2019 26th international conference on telecommunications. ICT, IEEE; 2019, p. 220–6.

[104] Kamble MR, Sailor HB, Patil HA, Li H. Advances in anti-spoofing: from the perspective of aSVspoof challenges. APSIPA Trans Signal Inf Process 2020;9:e2.

[105] Wu Z, Evans N, Kinnunen T, Yamagishi J, Alegre F, Li H. Spoofing and countermeasures for speaker verification: A survey. Speech Commun 2015;66:130–53.

[106] Zhang Y, Jiang F, Duan Z. One-class learning towards synthetic voice spoofing detection. IEEE Signal Process Lett 2021;28:937–41.

[107] Fang Y, Liu Y, Huang C, Liu L. FastEmbed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. Plos One 2020;15(2):e0228439.

[108] Hao J, Ho TK. Machine learning made easy: a review of scikit-learn package in python programming language. J Educat Behav Statist 2019;44(3):348–61.

[109] Zhang D, Chen H-D, Zulfiqar H, Yuan S-S, Huang Q-L, Zhang Z-Y, Deng K-J. iBLP: an XGBoost-based predictor for identifying bioluminescent proteins. Comput Math Methods Med 2021;2021:1–15.

[110] Reitz K. Requests: HTTP for Humans™, MMXVIX; 2023, https://requests.readthedocs.io/en/latest/.

[111] Lv X, Peng T, Tang J, He R, Hu X, Jiang M, Deng Z, Cao W. A mitmproxy-based dynamic vulnerability detection system for android applications. In: 2022 18th international conference on mobility, sensing and networking. MSN, IEEE; 2022, p. 408–16.

[112] Lee T, Singh VP, Cho KH, Lee T, Singh VP, Cho KH. Tensorflow and keras programming for deep learning. Deep Learn Hydrometeor Environ Sci 2021;151–62.

[113] Offensive-Security. Kali Linux | Penetration Testing and Ethical Hacking Linux Distribution. OffSec Services Limited; 2023, https://www.kali.org/.

[114] Canonical MS. Ubuntu: Enterprise open source and Linux. Canonical Ltd; 2023, https://ubuntu.com/.

[115] Rapid7. Metasploitable 2. Rapid7; 2023, https://docs.rapid7.com/metasploit/metasploitable-2/.

[116] Wood R. Damn vulnerable web application (DVWA). 2023, GitHub, https://github.com/digininja/DVWA.

[117] Canty R, Dyke J, Wilson A. Vulnado - Intentionally vulnerable Java application. 2020, GitHub, https://github.com/ScaleSec/vulnado.

[118] Wireshark-Foundation. Wireshark - the world's most popular network protocol analyzer. 2023, Wireshark Foundation, https://www.wireshark.org/.

[119] Gu H, Zhang J, Liu T, Hu M, Zhou J, Wei T, Chen M. DIAVA: a traffic-based framework for detection of SQL injection attacks and vulnerability analysis of leaked data. IEEE Trans Reliab 2019;69(1):188–202.

[120] Li Q, Li W, Wang J, Cheng M. A SQL injection detection method based on adaptive deep forest. IEEE Access 2019;7:145385–94.

[121] Xie X, Ren C, Fu Y, Xu J, Guo J. Sql injection detection for web applications based on elastic-pooling cnn. IEEE Access 2019;7:151475–81.

[122] Arasteh B, Aghaei B, Farzad B, Arasteh K, Kiani F, Torkamanian-Afshar M. Detecting SQL injection attacks by binary gray wolf optimizer and machine learning algorithms. Neural Comput Appl 2024;36(12):6771–92.

[123] Apruzzese G, Laskov P, Montes de Oca E, Mallouli W, Brdalo Rapa L, Grammatopoulos AV, Di Franco F. The role of machine learning in cybersecurity. Digit Threat: Res Practice 2023;4(1):1–38.

[124] Stricklandz E. Andrew ng: Unbiggen AI. IEEE Spectrum; 2022, https://spectrum.ieee.org/andrew-ng-data-centric-ai.
[125] Garrity P. Taking an evidence-based approach to vulnerability prioritization. 2024, VulnCheck, https://vulncheck.com/blog/vulnerability-prioritization.
[126] Madden C. Risk based prioritization. Material for MkDocs; 2024, https://riskbasedprioritization.github.io/.
[127] Hughes C. Death knell of the nvd?. 2024, Resilient Cyber, https://www.resilientcyber.io/p/death-knell-of-the-nvd.
[128] Vaughan-Nichols SJ. NVD slowdown leaves thousands of vulnerabilities without analysis data. The Register; 2024, https://www.theregister.com/2024/03/22/opinion_column_nist/.