Queen's University Belfast

School of Electronics, Electrical Engineering and Computer Science

ELE8095 Individual Research Project

Project Title: *SQL Injection Attack Detection and Prevention*

Student Name: Alan Paul

Student Number: 40228223

Academic Supervisor: Dr Oluwafemi Olukoya

11th September 2023

Gitlab Repository: [1]

Declaration of Academic Integrity

I declare that I have read the University guidelines on plagiarism – https://www.qub.ac.uk/directorates/AcademicStudentAffairs/AcademicAffairs/AppealsCo mplaintsandMisconduct/AcademicOffences/Student-Guide/[1] - and that this submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.

Student's Signature: Alan Paul *Date of submission: 11/09/2023*

Acknowledgements

I wish to convey my heartfelt appreciation to my supervisor, Dr Oluwafemi Olukoya. For the second consecutive year, his unwavering support, insightful feedback, and valuable guidance have been instrumental to my academic aspirations. I am grateful for the foundation he has laid in my academic pursuits, and for the privilege of learning under his mentorship for another year. I would also like to thank my family for their continuous encouragement and belief that propelled me forward this year. Once again, I would like to extend my gratitude to the CSIT engineering team, ECIT Institute for guiding me into Cyber Security during placement year. Their influence has been essential to both my academic and professional journey.

Table of Contents

Abstract

Context

Web applications have become central in the digital landscape, providing users instant access to information, and allowing businesses to expand their reach. However, with the number of websites reaching approximately 1.7 billion, it has also drawn the attention of threat actors [2]. SQL injection (SQLi) attacks remain common and will continue to persist as long as code is being written. Given that most web applications integrate a database system, there is a wide attack surface presented for threat actors.

Objectives

The project aimed to train both machine and deep learning models on real-world SQLi datasets for detecting attacks from various sources. Additionally, the project evaluates current SQLi attack detection solutions and proposes a holistic approach, SQLR34P3R, that builds upon existing methods by implementing previously unconsidered metrics.

Results

The dual model of LSTM and CNN-LSTM implemented in SQLR34P3R excelled in both web and network traffic filtering respectively. SQLR34P3R achieved a precision score of 99.8994%, recall of 99.8998%, an F1 score of 99.8994% and False Positive Rate (FPR) of 0.00020 for SQLi payload detection. For SQLi detection within network traffic, SQLR34P3R achieved a precision score of 96.0035%, recall of 99.4041%, F1 Score of 97.67% and FPR value of 0.0414. Moreover, SQLR34P3R incorporates a novel risk analysis approach which reduced additional effort while maintaining

reasonable coverage to assist businesses focus on patching vulnerabilities with high exploitability.

1. Introduction

Web applications are frequently targeted by threat actors due to their rapid production and wide attack surface. Recent data from Astra Security [3] states that a web application faces a cyber-attack every 39 seconds. OWASP's Top Ten lists SQLi among the top three web vulnerabilities in both the 2017 and 2021 versions [4]. The severe impacts of SQLi attacks, which have cost the US economy a loss of around $10 billion, underlines why SQLi detection and prevention have become a primary focus of research [5]. The surge in web applications and smart phones usage coupled with the increasing technical proficiency of end users has led to a spike in web traffic. Businesses migrating their systems from offline systems to online systems has provided threat actors with an increased attack surface to exploit [6].

1.1 Background

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 unauthorised operations. An example usage of SQLi attack to bypass authentication is demonstrated in Figure 1.

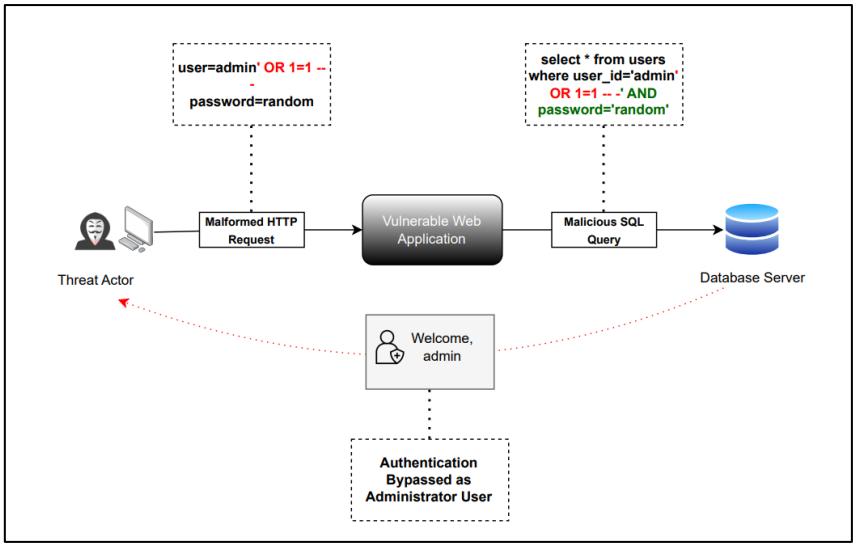


Figure 1 - SQL Injection Example

Figure 1 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. 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 Referrer) [7].

SQLi attacks can fall into three primary categories: Classic, Blind-based and Out-of-Band SQLi. Classic attacks such as Error and Union-based, confirm vulnerabilities through error messages in web responses or by combining queries of matching columns. Blind-based attacks such as Boolean or Time-based use conditional SQL queries and time delays to identify vulnerabilities. Out-of-Band uses non-standard protocols like DNS to retrieve data through alternative channels. [8]

1.2 Problem Statement

With the widespread accessibility of specialised 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. The impact of SQLi attacks on the CIA of data is explained in Table 1 [9].

Impact	**Description**
Compromise of 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.
Compromise of Integrity	Threat actors can use standard SQL commands such as "UPDATE" and "ALTER" to make changes to the database records.

Compromise of Availability	Threat actors are able to use commands such as “DELETE” and “DROP” to delete individual records or the entire table respectively.

Table 1 - Impacts of SQL Injection

Table 2 showcases notable SQLi attacks that occurred between 2021 and 2023 which had substantial impact on various entities. The table also includes the category of the application affected, attack classification, affected component and consequences.

Platform	**Category**	**Attack Type**	**Affected Component**	**Impact**
WooCommerce, 2021 [10]	WordPress Plugin	Authentication Bypass	Webhook Search Parameter	Data Theft
BQE BillQuick, 2021 [11]	Web Application	Remote Code Execution	“txtID” (username) parameter	Malware Deployment
Django, 2022 [12]	Web Application	Blind SQL Injection	Trunc and Extract Functions	CIA Compromise
MOVEit, 2023 [13]	Web Application	Classic SQL Injection	UserEngine Function	Data Exfiltration

Table 2 - SQLi Real World Impacts

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. Some examples of compounded attacks are outlined in Table 3 [14]:

Attack Type	**Description**
SQLi Denial of Service	Denial of Service through SQLi is caused by performing recursive SQL operations to overload the database servers rendering it unavailable and preventing users from performing database operations.

SQLi DNS Hijacking	Non-HTTP protocols such as DNS can be used to hide SQLi attacks by enclosing the malicious SQLi commands in DNS requests to avoid detection.
SQLi DNS Data Exfiltration [15]	SQLi types such as Blind SQLi attacks need to be combined with protocols such as DNS to exfiltrate data in a discreet manner.

Table 3 - SQLi Compound Attacks

1.3 Motivation

SQLi vulnerabilities allows unauthorized database access leading to potential data breaches and causing significant financial and legal impacts for businesses involved. Although numerous solutions SQLi attack detection and prevention solutions have been developed, this research addresses gaps identified in current solutions. It has been recognised 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 [7]. However, as threat actors now exploit non-HTTP protocols such as DNS for data exfiltration as described in Table 3, and through various components described in Table 2, theres a need for a complete inspection of all elements of a HTTP request including the query parameters, HTTP headers and request body as well as network layer information.
- **Multiclassification:** SQLi attacks can vary, from authentication bypass to remote code execution attacks as presented in Table 2. Although some solutions detect various SQLi attacks [16], none of the solutions were able to attribute the detected attack to a specific type. Instead, they labelled each detected instance as either malicious or benign.
- **Attack Prioritization:** Risk modelling through attack prioritisation helps organisations allocate resources effectively as all SQLi attack types does not have equal severity or exploitability. By targeting the most critical SQLi threats, businesses can respond faster and save valuable time.

1.4 Research Questions

Through an in-depth review of 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 multisource SQLi detection methods in mitigating attacks?

This research evaluated the effectiveness of multisource 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 in Tables 2 and 3.

RQ2. What are the advantages of using multiclassification-based machine learning models to detect and categorise detected SQLi payloads?

Training a machine learning model with multiclassification 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 reaching the backend DBMS systems.

1.5 Project Contribution

In this paper, SQLR34P3R is proposed, which implements a threefold approach to SQLi detection and prevention derived from Sections 1.3 and 1.4: *Multisource Detection*, *Attack Multiclassification* and *Attack Prioritization* incorporating threat

intelligence feeds. As of now, a solution with these features has not been proposed before in existing literature.

2. Literature Review

2.1 Traditional Frameworks

Early SQLi detection frameworks adopted static analysis approaches like pattern matching to detect SQLi attacks. AMNESIA was one of the earliest adopted SQLi detection and prevention solution which used hybrid analysis to detect SQLi [17]. 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 [18], which acts as proxy to filter malicious SQLi traffic in real-time and authenticate users. Similarly, a framework which implements a combination of BCRYPT hashing algorithm and Aho-Corasick pattern matching technique was employed [19]. Aho-Corasick pattern matching technique stores a predefined set of patterns within a trie data structure which the algorithm can traverse through during the searching phase to find matches of malicious SQLi attack signatures. SAFELI, 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 [20]. The main disadvantage of proposed static analysis based SQLi attack detection methods was their inability to detect unknown SQLi attacks [21].

2.2 Machine Learning-based Frameworks

In recent years, the popularity of using machine learning models to detect SQLi attacks have surged due to their effectiveness over traditional methods [21]. SQLIFIX uses clustering to identify similarities between different PHP and Java codebases collected from GitHub that are vulnerable to SQLi and automatically generates remediation recommendations [22]. 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 prioritisation [23]. An alternative solution combined feature extraction methods such as query trees and Fisher score with SVM classifier from the Waikato Environment for Knowledge

Analysis (Weka) library [24]. The query trees for both benign and malicious SQLi data is 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 [25]. 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 [26]. SQLBlock, a plugin designed for PHP and MySQL applications, detects SQLi vulnerabilities through hybrid analysis preventing execution of unsafe PHP functions [27].

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 [21]. 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 [28]. 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 [29]. 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 [30]. 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 [31].

A semantic learning-based detector known as synBERT was proposed which uses embedding vectors to recognise the patterns within individual SQL queries which uses syntax tree structures to understand the grammatical context within SQL statements [21]. A recent proposal utilised the benefits of deep learning by employing a

probabilistic neural network (PNN) to enhance SQLi attack detection accuracy [32]. The approach used 6000 SQLi queries as malicious dataset and 3500 benign queries alongside network traffic. A combination of regular expressions and tokenization were used for data preprocessing 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 4.

Solution	Advantages	Disadvantages
AMNESIA [17]	• Efficient at reducing false positives. • Does not need to interact with client or backend.	• Stored procedures attacks go undetected. • Specific to Java web apps.
Regular Expression pattern matching [18]	• Does not need to interact with client or backend.	• Stored procedures attacks go undetected. • Can be bypassed using advanced attacks.
Aho-Corasick pattern matching [19]	• Minimised false results (false positives and false negatives).	• Does not consider the overall SQL statement structure. • Can be bypassed using advanced attacks.
SAFELI [20]	• 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 [22]	• Language Agnostic. • Performs better than Prepared statement replacement algorithm (PSR)	• Unable to detect SQLi in 139 code segments. • Unable to detect SQLi in certain conditions: presence of certain SQL modifiers, query errors and lack of batch process support in prepared statements.
SVM with node centrality [23]	• Low performance overhead. • Safeguards multiple websites on deployed server.	• Solely focuses on WHERE clause values. • Lack of testing performed with diverse web apps.
SVM with Query Trees and Fisher Score [24]	• Distinguishes between normal users and threat actors. • Eliminates redundant features.	• SQL queries were limited in testing dataset. • Lack of detail on detectable SQLi attack types.
Multisource SQL Injection Detection [25]	• 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 [26]	• Handles large volumes of data efficiently.	• Only can be implemented with login endpoints. • Cannot detect sophisticated SQLi attacks. [33]
SQLBlock Plugin [27]	• 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 [28]	• Works well against classic SQLi attacks.	• Model not trained on advanced SQLi attacks.
Traditional CNN [29]	• Outperformed rule-based solution, Modsecurity in terms of evaluation metrics.	• Trains models with dataset containing limited types of SQLi attacks. • Does not perform sufficient comparative analysis with state-of-the-art.
LSTM [30]	• Mitigates overfitting and detects diverse SQLi attack types.	• Attack scenarios considered by the solution may not be comprehensive.
LSTM/MLP with advanced Feature Extraction [31]	• Enhances accuracy using a unique feature extraction implementation.	• Only evaluates input from URLs and overlooks alternative attack vectors.

synBERT [21]	• Solution offers higher flexibility and better detection rate for previously unknown attacks.	• Model is trained on limited dataset.
PNN [32]	• High accuracy with low performance overhead and false positive rate.	• Susceptible to interference from irrelevant features. • High complexity.

Table 4 - State-of-the-art Advantages and Disadvantages

2.5 Gap Analysis

By performing gap analysis, critical limitations have been identified, highlighting areas for potential improvements. In this project, multiclassification 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 4 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 [16] failed to specify the types of detected SQLi, highlighting the need for a multiclassification-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 4 consider only the HTTP protocol as the attack vector, ignoring other potential SQLi attack vectors such as DNS. By implementing multisource 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.

3. Project Methodology

Implementation of SQLi detection within 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, utilising both traditional machine learning and deep learning models.

3.1 Classification and Detection Framework

Figure 2 illustrates the detection framework comprised of four main components: Data gathering and aggregation, Data Preprocessing, Feature Extraction, Classification and Evaluation.

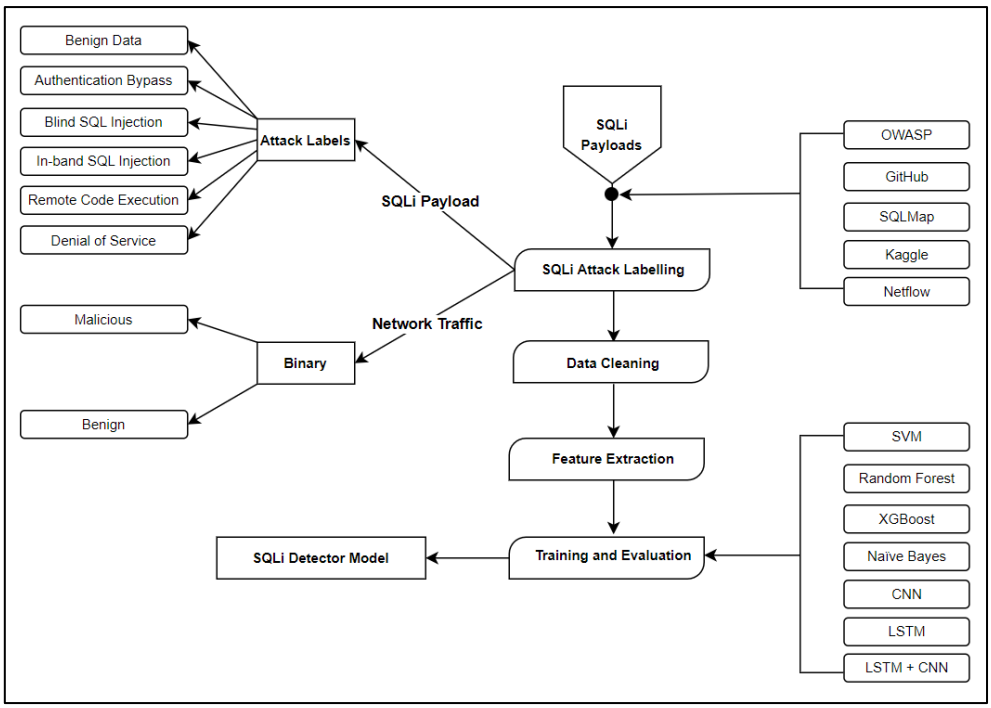


Figure 2 - Detection Framework

3.1.1 SQLi Attack Collection and Labelling

Multiclass payload samples that were aggregated from multiple sources are listed in Table 17 within Section 9.1 of the Appendix. Table 17 highlights the sample types and counts each source, covering both payload and NetFlow dataset compilation. For attack classes such as Authentication Bypass, Classic SQLi, Blind SQLi, and Remote Code Execution the corresponding SQLi payloads were already prelabelled by Github[34][35][36], SQLMap[37] and OWASP [38] respectively. The accuracy of these mappings and datasets from Kaggle[39] were further validated by performing literature

review where characteristics of each malicious SQL query type were established. The literature reviewed for validating each attack characteristic are detailed in Table 5. For the NetFlow dataset [40], each traffic flow was pre-labelled as either malicious (1) or benign (0), reflecting a binary classification problem. Consequently, there was no need for further categorization. Table 5 showcases examples of each attack type along with its corresponding label and literature used to perform payload characterisation.

Literature	Malicious SQLi Payload Characteristic	Attack Type	Label
[41]	alan' or '2'='2'-- -	Authentication Bypass	0
[41]	; WAITFOR TIME '18:15:00'; -- -	Blind SQL Injection	1
[14]	select derived_table.table1 from (select encode(decode(compress(convert(post) using latin1)),md5(concat(post,post))) as derived_table;	Denial of Service	2
[41]	' UNION SELECT 'a',NULL,NULL,NULL-- -	Classic SQL Injection	3
[42]	; exec master..xp_cmdshell 'certutil -urlcache -f https://redteam/'--	Remote Code Execution	4
[43]	SELECT DEPARTMENT, COUNT(*) FROM EMPLOYEES;	Normal SQL Operation	5

Table 5 - SQLi Attack Labelling

3.1.2 Data Pre-Processing and Data Cleaning

Data pre-processing consists of standardising data into a format suitable for the machine learning model. Any duplicate or empty entries that may have accumulated during edataset collection for the SQLi payload datasets were removed by executing bash scripts to improve dataset reliability. 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 [44].

3.1.3 Feature Extraction

Natural language processing (NLP) was performed on the SQLi payload dataset to convert the text based SQLi payloads into machine readable format. 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 [45]. 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. TF-IDF can be calculated using the formula stated below [45]:

$$\mathrm{TF} - \mathrm{IDF} = TF(w, P) \times IDF(w)$$

Based on the formula stated above, 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”.

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

Tokenization and Global Vectors for Word Representation (GloVe) is then applied to perform word embeddings for the CNN model to break down the SQLi payloads to tokens. GloVe is a word vectorizer technique that is used to contextualise and represent words in machine readable format [46]. 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 throughout the entirety of a document which facilitates a broader understanding of word relationships. As a result, GloVe can encode deeper relationships which allows it to capture nuanced information. The feature values from the NetFlow dataset were normalised 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 normalisation due to its speed, high scalability, and linearity [47]. Standard Scaler 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 [44].

3.1.4 Classification

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 5. After feature extraction, the combined dataset is the divided with 60% designated for training data and 40% designated for testing data to achieve a more accurate evaluation of the models' performance on diversified set of unseen data. Additionally, the samples were stratified due to the different number of individual datasets to prevent any bias arising from imbalanced datasets. Table 6 shows the class/label count in both testing and training divisions for SQLi payload dataset.

Training Split (60%)					
Authentication Bypass	*Blind SQL Injection*	*Denial of Service*	*Classic SQL Injection*	*Remote Code Execution*	*Benign Data*
149	9,623	192	9,284	33	22,626
Testing Split (40%)					
Authentication Bypass	*Blind SQL Injection*	*Denial of Service*	*Classic SQL Injection*	*Remote Code Execution*	*Benign Data*
99	6,416	128	6,190	21	15,084

Table 6 - SQLi Payload: Training & Testing Split

The NetFlow dataset was used for binary classification. Both the provided NetFlow datasets were combined to a single dataframe before being split for training and testing. The NetFlow datasets were split in a standard 70:30 ratio. Table 7 highlights the count of both classes in both testing and training divisions for NetFlow dataset.

Training Split (70%)

Malicious Netflow Data	*Benign Netflow Data*
159,985	160,077
Testing Split (30%)	
Malicious Netflow Data	*Benign Netflow Data*
68,631	68,539

Table 7 - NetFlow: Training & Testing Split

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 Network (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 [48]. Random Forest had the best results with an accuracy of 96.8% and 95.69% for both binary and multi-classification respectively, while Naïve Bayes had 91% and 81.8% and SVM also proved effective, achieving an accuracy of 88.6%, making it an appropriate model for handling both linear and non-linear relationships in classification tasks. LSTM in combination with GloVe embeddings improved performance consistently across metrics, even with an unbalanced dataset [49]. Combining GloVe word embeddings with both LSTM and CNN significantly reduces computation time, while maintaining high accuracy levels. Both CNN and LSTM individually achieve respectable accuracy, highlighting their robustness and flexibility [49]. The research suggests that both LSTM and CNN models are known for their efficiency, performance, and adaptability in both binary and multi-classification problems. However, even though CNNs can effectively capture local semantics between texts, they struggle to understand the wider context while LSTM models are time consuming to train. The combined CNN-LSTM hybrid model addresses these limitations by utilising CNN for feature extraction and LSTM for wider contextual understanding between text data.[50]

To evaluate the implemented models, standard evaluation metrics were utilised which consists of Accuracy, F1 Score, Precision, Recall and False Positive Rate (FPR). The calculations performed for each of the evaluation metrics are as follows [28][31]:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

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

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

$$\text{F1 Score} = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)}$$

$$\text{False Positive Rate} = \frac{FP}{FP + TN}$$

3.2 Experiments and Results

The machine learning and deep learning models were evaluated based on their ability to accurately identify and differentiate between 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**. 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.

The deep learning model parameters for the SQLi payloads are presented in Table 8.

Parameter	**Values**
Optimizer	Adam
Loss Function	Sparse Categorical Crossentropy
Epochs	10

Batch Size	64

Table 8 - Payload Deep Learning Model Parameters

The implemented deep learning model architectures for the SQLi payload data is illustrated in Figures 3 - 5.

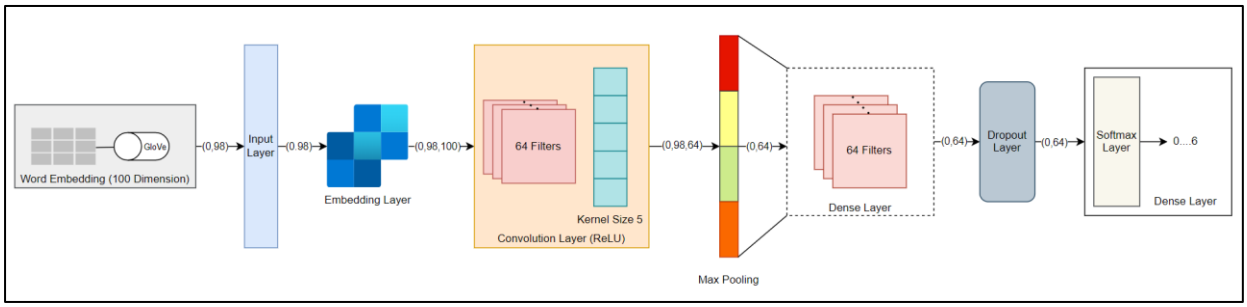


Figure 3 – Payload: CNN Architecture

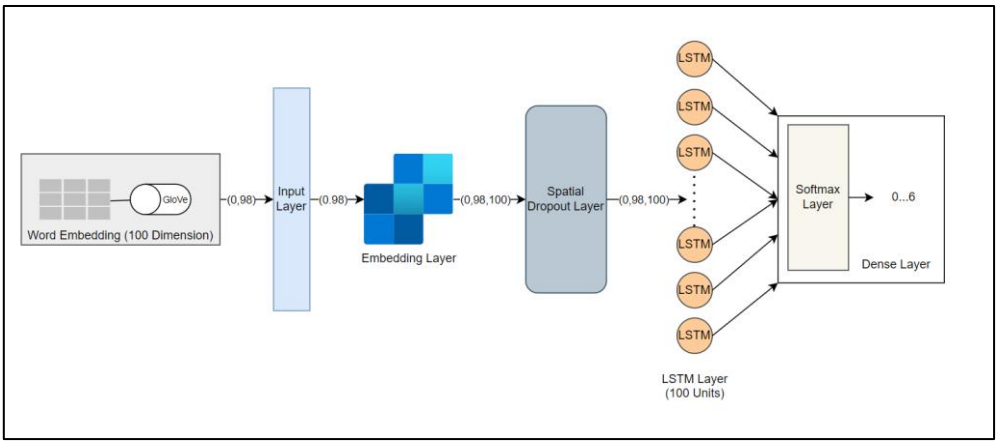


Figure 4 – Payload: LSTM Archtitecture

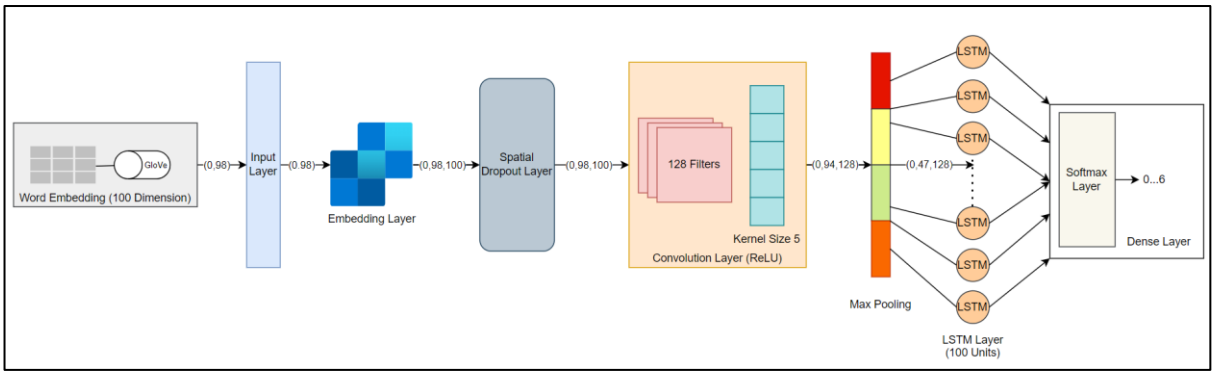


Figure 5 – Payload: CNN-LSTM Architecture

For the three deep learning model implementations, GloVe word embeddings is used to understand word semantics. For the CNN implementation, the 1D convolutional layer is used in combination 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 dropout layer is used to prevent overfitting. LSTM

architecture utilises a 100-unit LSTM layer to contextualise 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, the CNN is used for feature extraction which identifies and extracts short patterns from text. The max pooling then layer is then used to highlight key patterns and reduces the data size. Finally, LSTM layer is the implemented to understand the long-term dependences 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 minimum sample split of 0.1.
- **XGBoost**. 100 decision trees were combined with learning rate and L1 regularization of 0.1.
- **Naïve Bayes (NB).** Hyperparameter tuning had no beneficial impact on the default settings.

The deep learning model parameters for the NetFlow dataset are presented in Table 9.

Parameter	Values
Optimizer	Adam
Loss Function	Binary Crossentropy
Epochs	20
Batch Size	32

Table 9 - NetFlow Deep Learning Model Parameters

The implemented deep learning model architectures for the NetFlow data are illustrated in Figures 6 - 8.

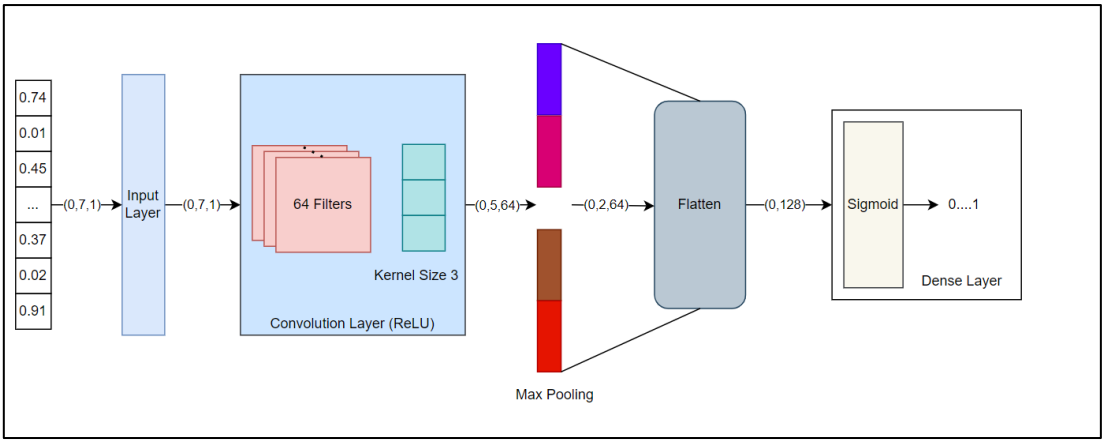


Figure 6 – NetFlow: CNN Architecture

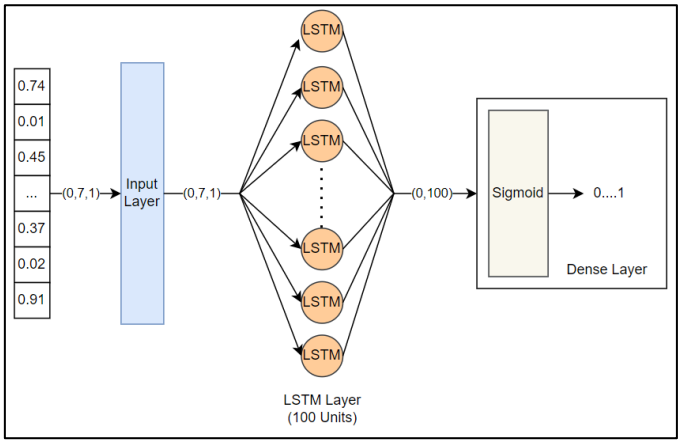


Figure 7 – NetFlow: LSTM Architecture

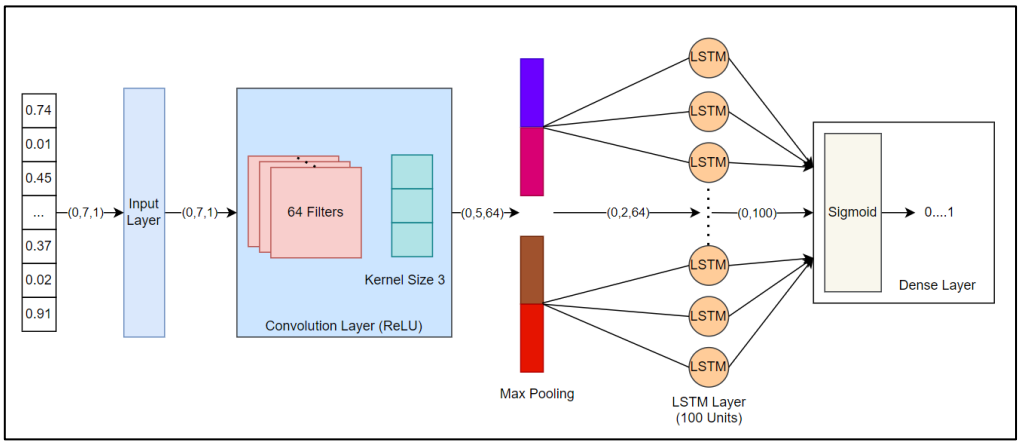


Figure 8 – NetFlow: CNN-LSTM Architecture

As shown in Figures 6-8, 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 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.

RQ1 and RQ2 aims to establish the benefits of utilizing multiclassification-based machine learning models for i) detecting SQL injection payloads and ii) categorizing the detected SQL injection types. To answer RQ1 and RQ2, the performance of the machine learning models specified in Section 3.1.4 were measured in terms of the evaluation metrics accuracy, precision, recall, F1 score, and FPR as well their ability to successfully identify and attribute attacks to their respective classes. Table 10 contains the results of the classifier evaluation, showing the overall performance of the models in detecting SQLi payloads. Table 18 in Appendix, Section 9.2 provides a detailed breakdown of the classifier performance for each individual class of SQLi attack.

	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 Score (%)**	**FPR**
SVM	99.5132%	99.5142%	99.5132%	99.5103%	0.00097
Random Forest	99.8497%	99.8489%	99.8497%	99.8477%	0.00030
XGBoost	99.7566%	99.7558%	99.7566%	99.7536%	0.00049
Naïve Bayes	96.5925%	95.7297%	96.5925%	96.1569%	0.00682

CNN	99.5741%	99.5620%	99.5741%	99.5149%	0.00085
LSTM	99.8998%	99.8994%	99.8998%	99.8994%	0.00020
CNN + LSTM	99.8855%	99.8885%	99.8855%	99.8854%	0.00023

Table 10 – SQLi ML Model Evaluation: Overall Results

The results from the classifier evaluation presented in Table 10 highlights that the deep learning models outperformed 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 in terms of higher accuracy and lower FPR, the LSTM model had the highest Accuracy value of 99.8998%, Precision value of 99.8994%, Recall value of 99.8998% and F1 score of 99.8994% while also having the lowest FPR of 0.00020. Although XGBoost was able to detect the Remote Code Execution and Denial of Service classes with a FPR of zero for the individual classes, the LSTM model was able to detect all classes with near perfect values for the individual evaluation metrics while XGBoost performed slightly worse in identifying blind-based SQLi attacks and benign data. The LSTM model outperforms the hybrid CNN+LSTM model due to its capability to handle imbalanced datasets effectively. This observation aligns with findings in existing literature where LSTM demonstrated better performance compared to the CNN-LSTM model in some scenarios [49]. The results from Table 10 confirm that the LSTM model had the best performance in identifying all the classes accurately for SQLi payload detection. To provide a comprehensive view of the performance of the two best performing models for the payload data, the progress has been visualised using graphs. Figure 9 and 10 illustrates loss and accuracy for the training and testing of data by LSTM. Figure 11 presents the confusion matrix.

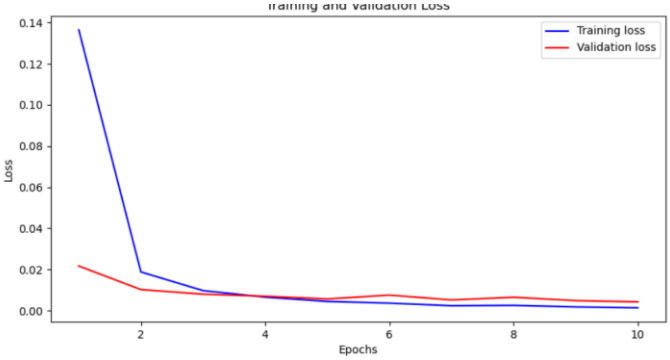


Figure 9 – LSTM: Training and Validation Loss

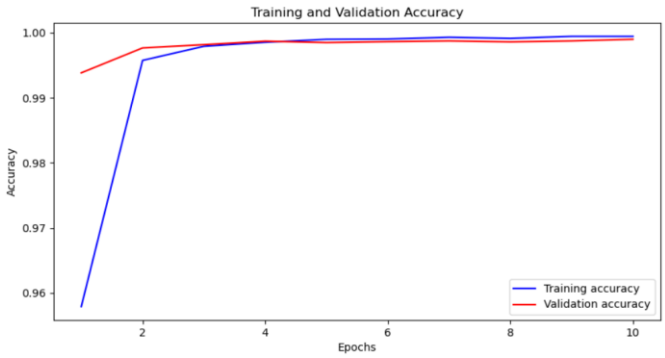


Figure 10 - LSTM: Training and Validation Accuracy

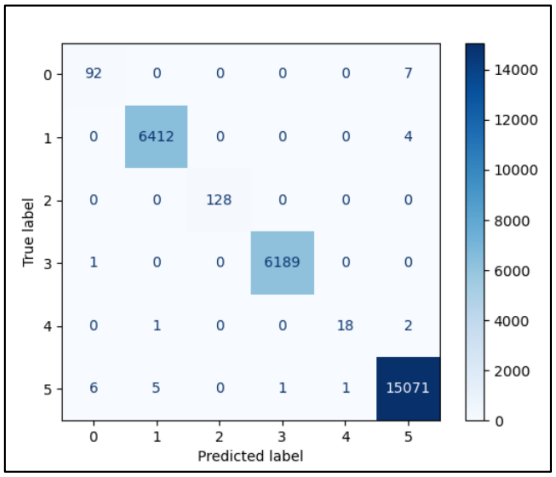


Figure 11 - LSTM: Confusion Matrix

Figure 12 illustrates the training and validation loss while Figure 13 illustrates the training and validation accuracy for the CNN-LSTM model implemented for the payload data. Figure 14 presents the confusion matrix.

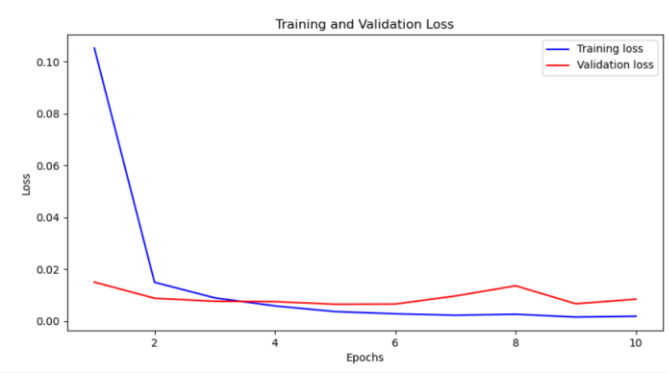


Figure 12 - CNN-LSTM: Training and Validation Loss

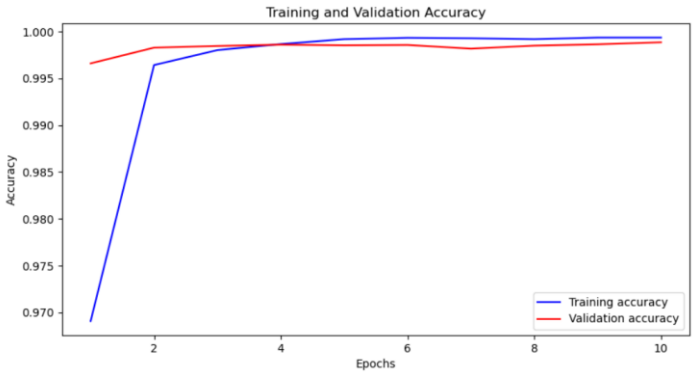


Figure 13 - CNN-LSTM: Training and Validation Accuracy

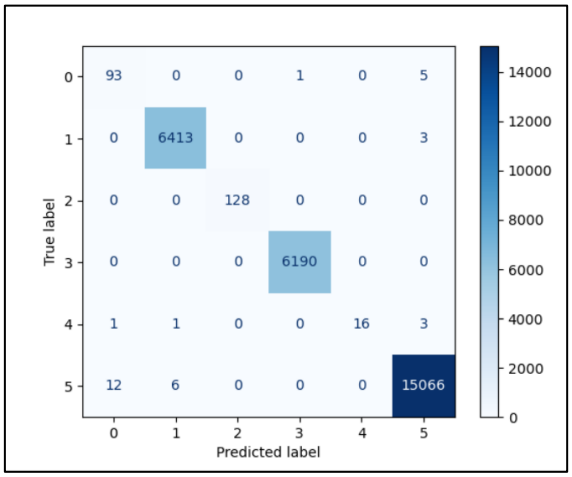


Figure 14 - CNN-LSTM: Confusion Matrix

Table 11 contains the results of the machine learning model evaluation for the NetFlow data, while Table 19 in Appendix, Section 9.2 provides a detailed breakdown of the ML model performance for the individual classes of the NetFlow dataset.

	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 Score (%)**	**FPR**
SVM	94.4777%	90.1589%	99.8630%	94.7632%	0.1091
Random Forest	85.9240%	78.8570%	98.1947%	87.4698%	0.2636
XGBoost	74.3275%	66.1165%	99.8718%	79.5619%	0.5125

Naïve Bayes	94.8691%	92.9106%	97.1587%	94.9872%	0.0742
CNN	94.8852%	90.8117%	99.8834%	95.1318%	0.1012
LSTM	61.6170%	99.6829%	23.3597%	37.8497%	0.0007
CNN + LSTM	97.6314%	96.0035%	99.4041%	97.6742%	0.0414

Table 11 - NetFlow ML Model Evaluation: Overall

For the NetFlow classification, the CNN and LSTM hybrid model outperformed the other ML and DL models with an overall accuracy, precision, recall and f1 scores of 97.63%, 96.00%, 99.40%, 97.67% respectively. Moreover, this model had a relatively low FPR of 0.0414. The balanced performance of the CNN and LSTM hybrid 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 consistent high performance across both overall and class specific evaluations makes it the most reliable model for the NetFlow dataset. Figure 15 illustrates the training and validation accuracy while Figure 16 illustrates the training and validation loss for CNN implemented for the NetFlow data. Figure 17 presents the confusion matrix.

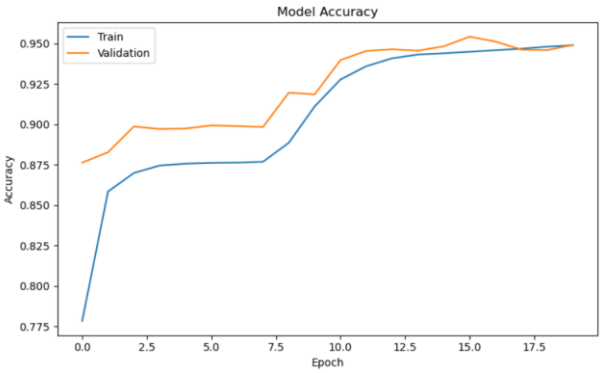


Figure 15 - CNN: Training and Validation Accuracy

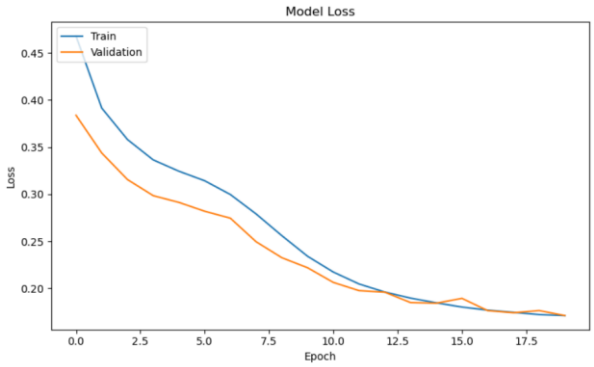


Figure 16 - CNN: Training and Validation Loss

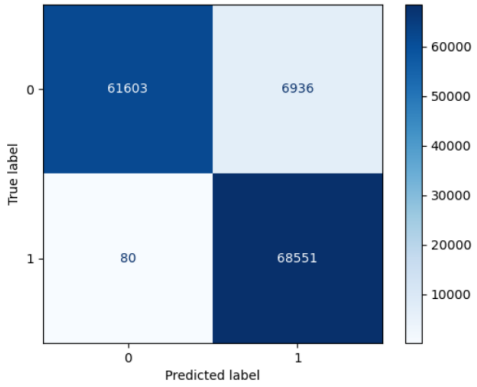


Figure 17 - CNN: Confusion Matrix

Figure 18 illustrates the training and validation accuracy while Figure 19 illustrates the training and validation loss for CNN-LSTM implemented for NetFlow data. Figure 20 presents the confusion matrix.

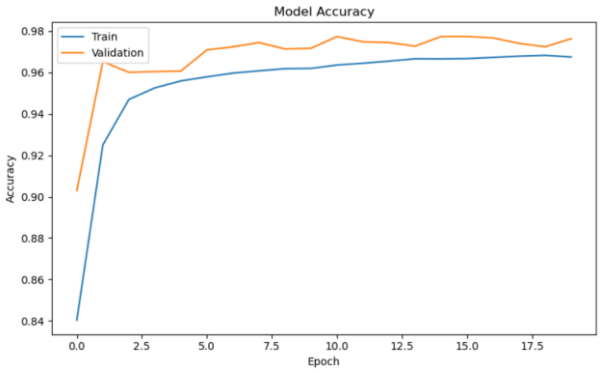


Figure 18 - CNN-LSTM: Training and Validation Accuracy

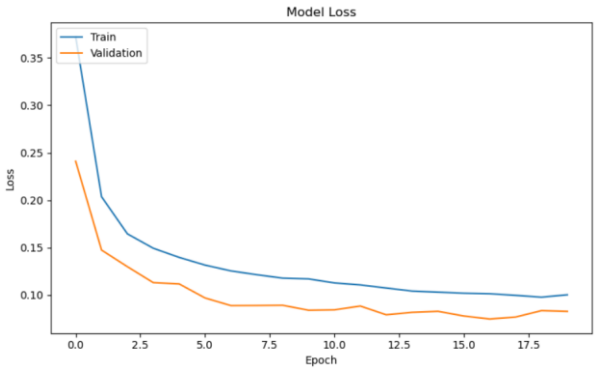


Figure 19 - CNN-LSTM: Training and Validation Loss

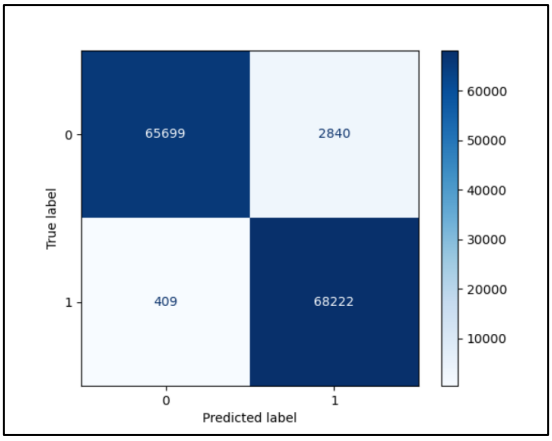


Figure 20 - CNN-LSTM: Confusion Matrix

3.3 Risk Analysis and Threat Modelling

Risk analysis was performed on the different SQLi attack classes to prioritise their risks. In addressing RQ3, this research aimed to model risks based on the identified SQLi classifications to aid efficient attack prioritisation. 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 for the risk analysis phase.

3.3.1 Common Vulnerability Scoring System (CVSSv3)

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 [51]. 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. 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. Although solutions such as VPR are better alternatives than relying solely on CVSS, these solutions lack transparency and cannot be adopted by businesses or organisations due to closed-source nature of the system highlighting the need for open-source prioritization systems. [52]

3.3.2 Exploit Prediction Scoring System (EPSS)

The implementation of Exploit Prediction Scoring System (EPSS) addressed the limitations of CVSS by accurately forecasting the exploitability of vulnerabilities in the wild. 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. [52]

3.3.3 Expected Exploitability

Expected Exploitability (EE) is a similar approach to EPSS in that the two approaches are used to determine the active exploitation of the vulnerability in the wild. Furthermore, both EE and EPSS recognise 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 assess 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.[53]

3.3.4 OWASP Risk Rating Methodology

The OWASP Risk Rating Methodology (RRM) was established to calculate the severity of the vulnerability. OWASP RRM uses two main components to calculate the severity: likelihood and impact. This is calculated using the following formula [54]:

$$Risk = Likelihood \times Impact$$

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 [54]. 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 [55].

3.3.5 Context-Aware Vulnerability Prioritization (CAVP)

CAVP (Context-Aware Vulnerability Prioritization) is a recent solution that prioritizes risks based on an organisation's assets. Moreover, CAVP verifies the vulnerabilities with trusted sources and performs environmental scanning to identify the CVEs that will affect the organisations technological environment. CAVP supplements the original CVSS approach by combining the 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. [56]

3.3.6 RISK R34P3R

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 as mentioned in Sections 3.3.1 and 3.3.5 [52]. 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 focus on the likelihood of exploitation of the vulnerability [57]. Moreover, EPSS and EE have distinct features which sets them apart from each other. According to [53][57][58], EE uses public data sources while EPSS uses both private and public data sources. 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. CVEs were collected from the National Vulnerability Database (NVD) spanning the years 2020 – 2023. For categorisation, 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 by combining the CVSS, EPSS and EE metrics.[54]

$$Risk\ Severity = AVG(N(EPSS)\ N(EE)) \times N(CVSS)$$

CVSS will be considered as an impact measure as CVSS measures how the vulnerability impacts the confidentiality, integrity, and availability of data [52]. Both EPSS and EE will form the likelihood measure as they both calculate the likelihood of vulnerabilities being exploited [53][58]. EPSS and EE generate outputs as values

between 0 and 1, so the output values will need to be normalised using the Min-Max formula [59]:

$$NV = \frac{V - min}{\max - min} * 9$$

To determine the severity of the risk, both the likelihood metrics (EPSS, EE) and impact metrics (CVSS) was calculated separately and was given an individual level based on Table 12 specified by OWASP [54]:

Likelihood/Impact Rating	
0 - 2	**Low**
3 - 5	**Medium**
6 – 9	**High**

Table 12 - Likelihood/Impact Rating Ranges [52]

After determining the likelihood and impact rating from Table 12, the overall severity rating can be computed using the overall risk rating table from OWASP RRM detailed in Table 13 [54].

Severity				
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Informational	Low	Medium
		Low	**Medium**	**High**
	Likelihood			

Table 13 - OWASP RRM Overall Risk Rating

Table 20 in the Appendix, Section 9.3 presents the CVE identifiers along with their corresponding CVSS, EPSS and EE scores, alongside RISK R34P3R risk ratings. Each business has different capacity to address application vulnerabilities, with resource availability playing a key role. Smaller companies with limited budget and resources, might prioritize less “effort” by choosing to address the most critical vulnerabilities without achieving complete “coverage” of all the vulnerabilities. “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 does 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 scores ≥ 0.022. Consider CVE-2023-36284 from Table 202: Its CVSS score of 7.5 is over the CVSS remediation threshold of 7.0, but both EPSS and EE scores fall under their 0.022 and 0.50 thresholds. 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 increases the additional effort required for patching if the vulnerability falls under coverage scope. Table 14 provides the coverage and additional effort required to remediate the vulnerability for each attack class using CVSS, EPSS, EE and RISK R34P3R.

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 SQLi		
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 14 - Vulnerability Coverage and Additional Effort

According to Table 14, RISK R34P3R stands out as the most effective vulnerability prioritisation 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, 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 a 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 business focus on patching vulnerabilities that are actively being exploited that can cause critical impact. Table 21 in Section 9.3 of the Appendix presents the average CVSS, EPSS, EE and

RISK R34P3R scores. Both the “Authentication Bypass” and “Remote Code Execution” SQLi classes have a CVSS severity of “Critical”. However, both the 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, RISK R34P3R model assigns them risk severities of “High” and “Medium” respectively after combining the CVSS, EPSS and EE scores.

3.3.7 Threat Modelling

To answer the second half of RQ3, this section combines the risk analysis process introduced above with intelligence gathering. This research recognises 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.

A. AbuseIPDB

AbuseIPDB 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. [60]

B. AlienVault Open Threat Exchange (OTX)

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.[61]

C. Shodan

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.[62]

4. System Design and Implementation

4.1 SQLR34P3R Overview

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 Figure 21. The trained CNN-LSTM model is deployed within the proxy as an integral part of the detection and multiclassification feature.

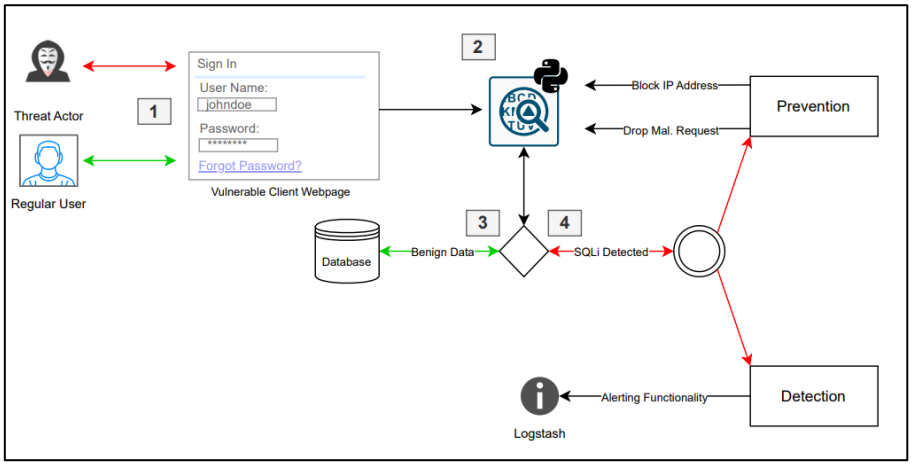


Figure 21 - SQLR34P3R Process Flow

In Figure 21, the regular user performs a legitimate login while the threat actor injects 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 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 appropriate response is shown to the authorised user. However, if SQLR34P3R detects SQLi payload instances, the instances are categorised to its respective attack labels based on the characteristics specified in Table 5 in Section 3.1.1. 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 of them such attacks are prohibited. Additionally, SQLR34P3R generates a risk rating using a novel vulnerability prioritisation approach, RISK R34P3R, combining severity, exploitability, and exploit availability metrics. Threat Modelling is also performed by determining the source of attack and by querying popular threat intelligence feeds to present the threat actor source, technique, system information. Figure 22 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 SQLi attack.

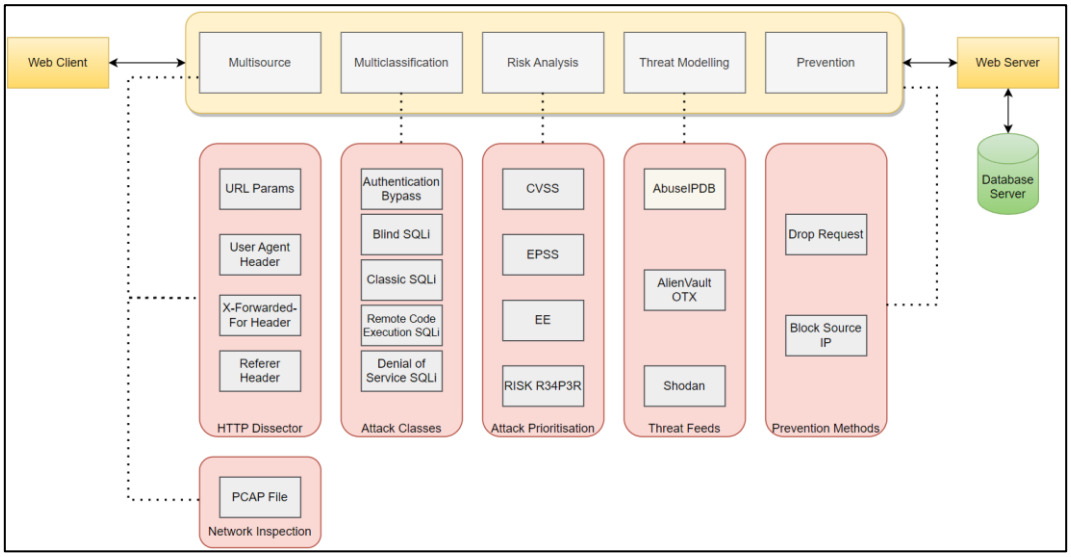


Figure 22 - Component Diagram

The state diagram displayed in Figure 23 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. As mentioned in Section 4.1, SQLR34P3R processes incoming web requests or inputted PCAP file 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.

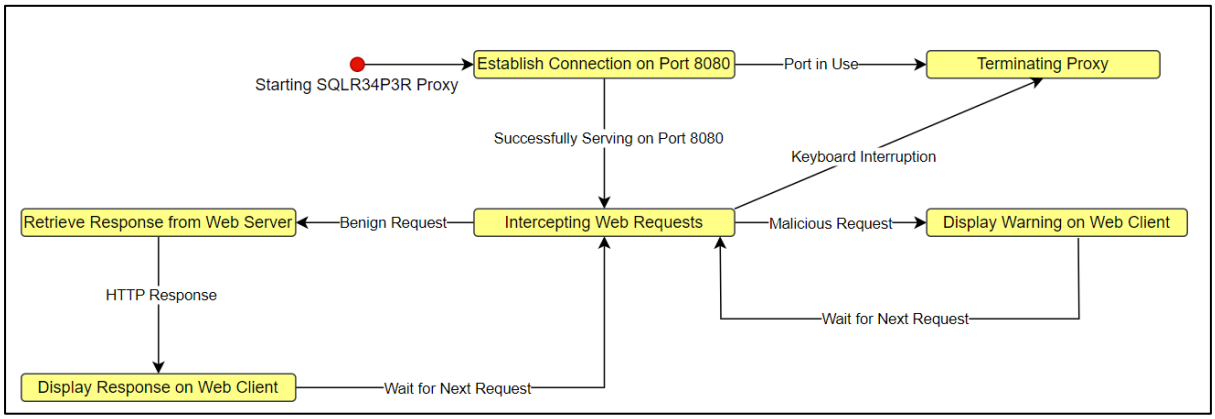


Figure 23 - State Diagram

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 Figure 24.

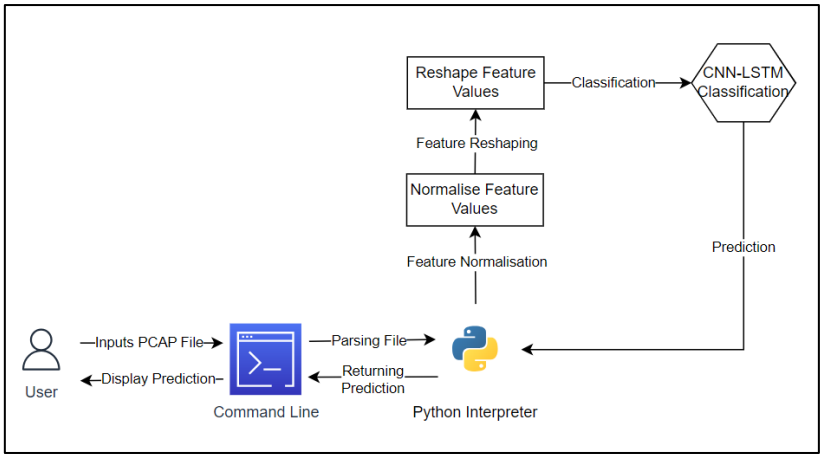


Figure 24 - Model Deployment

4.2 System Requirements

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

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

4.3 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 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 in Table 15.

Python Library	Description
Sklearn	Sklearn or Scikit-learn is a comprehensive and community driven machine learning library. Sklearn offers plethora of machine learning functionality including SVM, Random Forest to name a few. [63]
Xgboost	"xgboost" is a Python package which allows users to easily integrate XGBoost algorithm into their program. [64]
Requests	Requests is a popular Python package used to handle HTTP related tasks. It is possible to set custom headers, request and body parameters using Requests. [65]
Pandas	Pandas is a data analysis and visualisation library which makes it easy to work with structured data [63].
Mitmproxy	Mitmproxy library is the Python integration for the Mitmproxy proxy which can intercept HTTP traffic and perform application layer packet analysis.[66]
Tensorflow-Keras	Tensorflow combines with Keras API to simplify the process of developing and training deep learning models.[67]

Table 15 - Python Libraries

4.4 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”, “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, protocol type. The network layer information is then processed by the CNN-LSTM model to detect potential SQLi DNS data exfiltration.

4.5 Multiclassification and Categorisation

Datasets collected from sources specified in Table 6 is aggregated and separated into six different text files influenced by literature and the unique characteristics established in Table 5. 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 undergoes data preprocessing and feature extraction as described in Section 3.1.2 and 3.1.3. 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.

4.6 Risk Analysis Implementation

SQLR34P3R’s attack prioritisation 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.

4.7 Threat Intelligence Integration

As detailed in Section 3.3.7, three threat intelligence feeds were integrated into SQLR34P3R to perform threat modelling: AbuseIPDB, Alienvault OTX, Shodan. To incorporate AbuseIPDB, it’s 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 utilised to identify open ports and vulnerabilities.

4.8 Prevention System

To address RQ4, the implemented active response and prevention feature will immediately return 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.

5. System Demonstration and Contribution

5.1 Lab Environment Setup

The lab setup for the system demonstration which includes the software and network configuration is provided in Table 16. 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.

Software	Description	IP Address:Port
Kali Linux (2022.3)	Kali Linux is the hoist running the SQLR34P3R proxy.	10.0.2.4:80
Ubuntu (12.04)	The SQLi attacks will be initiated from the Ubuntu system posing as the threat actor.	10.0.2.15:80
DVWA (Metasploitable 2)	DVWA is an intentionally vulnerable PHP application hosted on Metasploitable2.	10.0.2.14:80

Vulnado	Vulnado is an intentionally vulnerable Java application hosted on Docker.	10.0.2.4:1337

Table 16 - Lab Environment Setup

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() -- -. Figure 25 confirms that DVWA is susceptible to SQLi attacks.

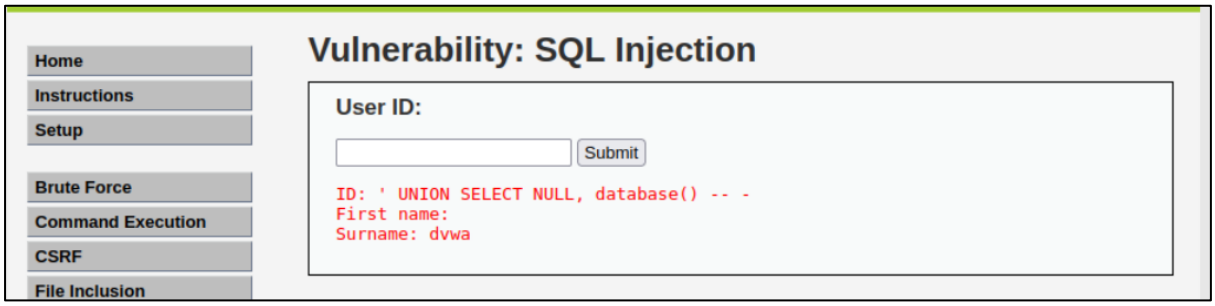


Figure 25 - DVWA SQLi Vulnerability

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”.

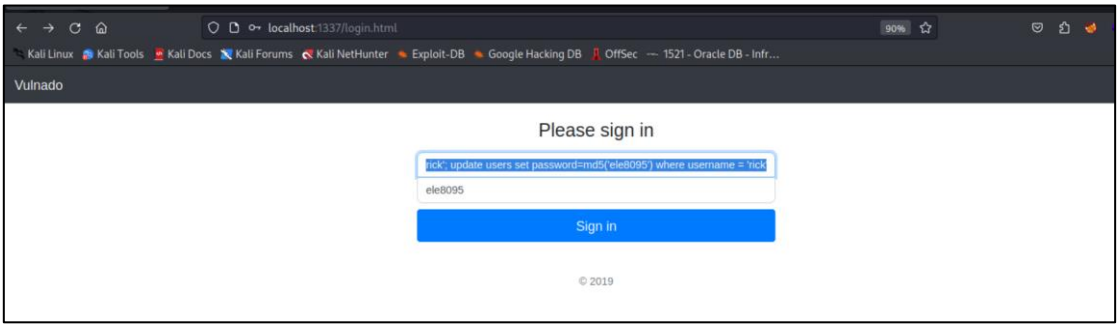


Figure 26 - Vulnado Vulnerable Endpoint

After executing the payload shown in Figure 26, the threat actor can bypass authentication by entering the updated password for the rick user and successfully login to the application as shown in Figure 27.

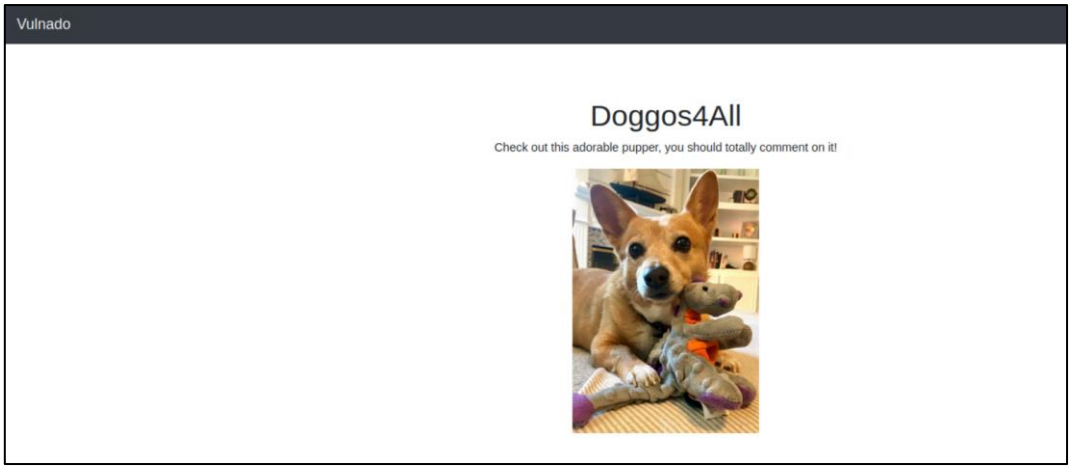


Figure 27 - Authentication Bypassed

5.2 Demonstration: SQLi Attack Detection, Multiclassification 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 individual classes. In Appendix Section 9.4, the successful detection and classification of various SQLi attacks using specific payloads are demonstrated:

- **Classic SQLi Attack:**
  - Figures 28 and 29 use payloads [68][69]:
    1. "*'+OR+1+GROUP+BY+CONCAT_WS(0x3a,VERSION(),FLOOR(RAND(0)*2))+HAVING+MIN(0)+OR+1-- -1693407438875*"
    2. "*'UnIoN SeLeCt CoUnT(TeXt) FrOm test.news WhErE 1=1 GrOuP By CoNcAt(VeRsIoN(),FlOoR(RaNd(1337)*2))*".
- **Authentication Bypass SQLi Attack:**
  - Figures 30 and 31 use payloads:
    1. "*admin))(|(|--*"
    2. "*') or ('')=('*".
- **Remote Code Execution SQLi Attack:**
  - Figures 32 and 33 use payloads:

1. "*EXEC master.dbo.xp_cmdshell 'powershell.exe dir c:/Users/Administrator*"'
2. "*EXEC xp_cmdshell 'ping <collab_url>.burpcollaborator.net'--*".

- **Blind SQLi Attack:**
  - Figures 34 and 35 use payloads:
    1. "*1);waitfor delay '0:0:10'--*"'
    2. "*1 AND (SELECT 1337 FROM (SELECT(SLEEP(5)))YYYY) AND (1337=1337*".
- **Denial-of-Service SQLi Attack:**
  - Figures 36 and 37 use payloads:
    1. "*1;SELECT BENCHMARK(5000000,MD5(0x6f537a66))*"
    2. "*2;SELECT BENCHMARK(5000000,SHA1(0x6f537a66))"*

In Figures 28-37 of Appendix Section 9.4, 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. Figure 38, in Section 9.5 of the Appendix, 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, Figure 39, in Section 9.6 of the Appendix, also displays the complete SQLR34P3R output, combining the detection/classification, risk analysis, and threat modelling results.

5.3 Demonstration: Multisource (DNS Exfiltration and Header Injection)

To demonstrate DNS data exfiltration during SQLi attacks via NetFlow data, network traffic was captured using Wireshark from SQLi DNS exfiltration conducted with sqlmap. Standard DNS traffic was also captured. Both datasets were saved as PCAP files for analysis. Figure 40 in Appendix Section 9.7 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 Figure 41. Additionally, SQLR34P3R detects SQLi attacks through vectors like User-Agent, Referer and X-Forwarded-For headers. Figures 42-

44 in Appendix Section 9.7 demonstrates the blind SQLi-based header injection. To differentiate between each header injection, each payload within the headers have been assigned a unique time element in the “waitfor” payload. Figures 45-47 in Section 9.7 of the Appendix demonstrate the successful detection and multiclassification of the SQLi attacks.

5.4 Demonstration: Prevention (Request Filtering through IP Blocking)

In Section 9.8 of the Appendix, Figure 48 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.

5.5 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 22 within Section 9.9 of the Appendix. SQLR34P3R presents a holistic approach when compared with current state-of-the-art solutions listed. Majority 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 its 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 38% 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.

6. Future Work and Limitations

SQLR34P3R currently analyses only HTTP requests for SQLi attack detection which will be further enhanced by inspecting HTTP responses. Some of the SQLi attack classes are hindered by limited datasets, therefore broader data collection will be performed from diversified trusted sources. Due to the unavailability of multiclass NetFlow datasets, it was not possible to implement multiclass detection for the network

traffic based SQLi attack detection. Additionally, the models for both the SQLi payload and network traffic were deployed individually due to different dataset structures preventing the development of a correlated model.

7. Conclusion

This research proposes SQLR34P3R, a command-line proxy that emerges as an essential solution in the domain of SQLi detection and prevention offering real-time web filtering and PCAP analysis capabilities for SQLi DNS data exfiltration. Its evident advantages over existing state-of-the-art approaches are owed to its three-pronged approach of multisource detection, multiclassification and comprehensive risk assessment highlighting its potential as an invaluable mechanism against SQLi attacks.

8. References

[1] Paul, A.: SQL Injection Attack Detection and Prevention. Gitlab (2022). https://gitlab2.eeecs.qub.ac.uk/40228223/sql-injection-attack-detection-and-prevention

[2] F. M. Alotaibi and V. G. Vassilakis, “Toward an SDN-Based Web Application Firewall: Defending against SQL Injection Attacks,” Future Internet, vol. 15, no. 5, p. 170, Apr. 2023, doi: https://doi.org/10.3390/fi15050170.

[3] N. James, “160 Cybersecurity Statistics: Updated Report 2023,” Astra, Dec. 19, 2022. https://www.getastra.com/blog/security-audit/cyber-security-statistics/ (accessed Jun. 01, 2023).

[4] OWASP, “OWASP Top Ten,” Owasp.org, 2021. https://owasp.org/www-project-top-ten/ (accessed Jun. 01, 2023).

[5] W. Zhang et al., “Deep Neural Network-Based SQL Injection Detection Method,” vol. 2022, Mar. 2022, doi: https://doi.org/10.1155/2022/4836289.

[6] J. Irungu, S. Graham, and A. Girma, “Artificial Intelligence Techniques for SQL Injection Attack Detection,” presented at the ICIIT ’23: Proceedings of the 2023 8th International Conference on Intelligent Information Technology, T. Kacem, Ed., Feb. 2023, pp. 38–45.

[7] Z. Zhu , S. Jia, J. Li, S. Qin, and H. Guo, “SQL Injection Attack Detection Framework Based on HTTP Traffic,” presented at the ACM TURC ’21:

Proceedings of the ACM Turing Award Celebration Conference, Oct. 2021, p. 181. Accessed: Jun. 03, 2023. [Online]. Available: https://doi.org/10.1145/3472634.3474068

[8] J. Padma N, Dr. R. N., Dr. R. M. B., and R. N.Ch., “SURGICAL STRIKING SQL INJECTION ATTACKS USING LSTM,” Indian Journal of Computer Science and Engineering, vol. 13, no. 1, p. 210, Feb. 2022, doi: https://doi.org/10.21817/indjcse/2022/v13i1/221301182.

[9] M. Abu Kausar, M. Nasar, and A. Moyaid, “SQL Injection Detection and Prevention Techniques in ASP.NET Web Application,” International Journal of Recent Technology and Engineering, vol. 8, no. 3, pp. 7759–7765, Sep. 2019, doi: https://doi.org/10.35940/ijrte.c6319.098319.

[10] I. Ilascu, “WooCommerce fixes vulnerability exposing 5 million sites to data theft,” BleepingComputer, Jul. 15, 2021. https://www.bleepingcomputer.com/news/security/woocommerce-fixes-vulnerability-exposing-5-million-sites-to-data-theft/ (accessed Jun. 04, 2023).

[11] C. Cimpanu, “Hackers use SQL injection bug in BillQuick billing app to deploy ransomware,” therecord.media, Oct. 25, 2021. https://therecord.media/hackers-use-sql-injection-bug-in-billquick-billing-app-to-deploy-ransomware (accessed Jun. 04, 2023).

[12] A. Sharma, “Django fixes SQL Injection vulnerability in new releases,” BleepingComputer, Jul. 04, 2022. https://www.bleepingcomputer.com/news/security/django-fixes-sql-injection-vulnerability-in-new-releases/ (accessed Jun. 04, 2023).

[13] Z. Hanley, “MOVEit Transfer CVE-2023-34362 Deep Dive and Indicators of Compromise,” Horizon3.ai, Jun. 09, 2023. https://www.horizon3.ai/moveit-transfer-cve-2023-34362-deep-dive-and-indicators-of-compromise/ (accessed Jun. 04, 2023).

[14] Z. S. Alwan and M. F. Younis, “Detection and Prevention of SQL Injection Attack: A Survey,” International Journal of Computer Science and Mobile Computing, vol. 6, no. 8, pp. 5–17, Aug. 2017.

[15] Miroslav Stampar, “Data Retrieval over DNS in SQL Injection Attacks,” arXiv (Cornell University), pp. 1–7, Mar. 2013, doi: https://doi.org/10.48550/arxiv.1303.3047.

[16] M. Nasereddin, A. ALKhamaiseh, M. Qasaimeh, and R. Al-Qassas, “A systematic review of detection and prevention techniques of SQL injection attacks,” Information Security Journal: A Global Perspective, vol. 32, no. 4, pp. 1–14, Oct. 2021, doi: https://doi.org/10.1080/19393555.2021.1995537.

[17] W. G. J. Halfond and A. Orso, “Preventing SQL injection attacks using AMNESIA,” presented at the 28th International Conference on Software Engineering (ICSE 2006), May 2006, pp. 1–4. Accessed: Jun. 07, 2023. [Online]. Available: https://www.cc.gatech.edu/home/orso/papers/halfond.orso.ICSEDEMO06.pdf

[18] A. Hadabi, E. Elsamani, A. Abdallah, and R. Elhabob, “An Efficient Model to Detect and Prevent SQL Injection Attack,” Journal of Karary University for Engineering and Science, pp. 1–5, Mar. 2022, doi: https://doi.org/10.54388/jkues.v1i2.141.

[19] S. Kini, A. P. Patil, M. Pooja, and A. Balasubramanyam, “SQL Injection Detection and Prevention using Aho-Corasick Pattern Matching Algorithm,” presented at the 2022 3rd International Conference for Emerging Technology (INCET), IEEE, 2022, pp. 1–6.

[20] X. Fu and K. Qian, “SAFELI: SQL injection scanner using symbolic execution,” presented at the Proceedings of the 2008 Workshop on Testing, Analysis, and Verification of Web Services and Applications, held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), Jul. 2008, pp. 34–39.

[21] D. Lu, J. Fei, and L. Liu, “A Semantic Learning-Based SQL Injection Attack Detection Technology,” Electronics, vol. 12, no. 6, p. 1344, Mar. 2023, doi: https://doi.org/10.3390/electronics12061344.

[22] M. L. Siddiq, Md. R. R. Jahin, M. R. Ul Islam, R. Shahriyar, and A. Iqbal, “SQLIFIX: Learning Based Approach to Fix SQL Injection Vulnerabilities in Source Code,” in IEEE Xplore, Mar. 2021, pp. 354–364. doi: https://doi.org/10.1109/SANER50967.2021.00040.

[23] D. Kar, A. K. Sahoo, K. Agarwal, S. Panigrahi, and M. Das, “Learning to detect SQLIA using node centrality with feature selection,” presented at the 2016 International Conference on Computing, Analytics and Security Trends (CAST), IEEE, 2017, pp. 18–23.

[24] A. Ladole and D. A. Phalke, “SQL Injection Attack and User Behavior Detection by Using Query Tree, Fisher Score and SVM Classification,” International Research Journal of Engineering and Technology (IRJET) , vol. 3, no. 6, pp. 1505–1508, 2016, Accessed: Jun. 08, 2023. [Online]. Available: https://www.irjet.net/archives/V3/i6/IRJET-V3I6283.pdf.

[25] K. Ross, M. Moh, T.-S. Moh, and J. Yao, “Multi-source data analysis and evaluation of machine learning techniques for SQL injection detection,” presented at the ACMSE ’18: Proceedings of the ACMSE 2018 Conference, Mar. 2018, pp. 1–8. doi: https://doi.org/10.1145/3190645.3190670.

[26] T. P. Latchoumi, M. Sahit Reddy, and K. Balamurugan, “Applied Machine Learning Predictive Analytics to SQL Injection Attack Detection and Prevention,” European Journal of Molecular & Clinical Medicine, vol. 7, no. 2, pp. 3543–3553, 2020.

[27] R. Jahanshahi, A. Doupé , and M. Egele, “You shall not pass: Mitigating SQL Injection Attacks on Legacy Web Applications,” presented at the ASIA CCS ’20: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, Oct. 2020, pp. 445–457.

[28] D. Chen, Q. Yan, C. Wu, and J. Zhao, “SQL Injection Attack Detection and Prevention Techniques Using Deep Learning,” Journal of Physics: Conference Series, vol. 1757, no. 1, pp. 1–8, Jan. 2021, doi: https://doi.org/10.1088/1742-6596/1757/1/012055.

[29] A. Luo, W. Huang, and W. Fan, “A CNN-based Approach to the Detection of SQL Injection Attacks,” presented at the 2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS), IEEE, 2019, pp. 320–324.

[30] Q. Li, F. Wang, J. Wang, and W. Li, “LSTM-Based SQL Injection Detection Method for Intelligent Transportation System,” IEEE Transactions on Vehicular Technology, vol. 68, no. 5, pp. 4186–4187, May 2019, doi: https://doi.org/10.1109/TVT.2019.2893675.

[31] P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu, “Detection of SQL injection based on artificial neural network,” Knowledge-Based Systems, vol. 190, pp. 1–10, Feb. 2020, doi: https://doi.org/10.1016/j.knosys.2020.105528.

[32] F. K. Alarfaj and N. A. Khan, “Enhancing the Performance of SQL Injection Attack Detection through Probabilistic Neural Networks,” Applied Sciences, vol. 13, no. 7, p. 4365, Jan. 2023, doi: https://doi.org/10.3390/app13074365.

[33] M. Qbea’h, S. Alrabaee, M. Alshraideh, and K. E. Sabri, “Diverse Approaches Have Been Presented To Mitigate SQL Injection Attack, But It Is Still Alive: A Review,” presented at the 2022 International Conference on Computer and Applications (ICCA), IEEE, 2023, pp. 2–3.

[34] Shala A. (2020), “sqli-auth-bypass.txt,” [GitHub]. https://gist.github.com/spenkk/2cd2f7eeb9cac92dd550855e522c558f

[35] Taşdelen I. (2021), “sql injection payload list,” [GitHub]. https://github.com/payloadbox/sql-injection-payload-list/blob/master/Intruder/exploit/Auth_Bypass.txt

[36] Polop C. (2021), “login-bypass,” [GitHub]. https://github.com/carlospolop/hacktricks/blob/master/pentesting-web/login-bypass/sql-login-bypass.md

[37] B. Damele and M. Stampar, “sqlmap: automatic SQL injection and database takeover tool,” sqlmap.org. https://sqlmap.org/

[38] “WSTG - Stable | OWASP Foundation,” owasp.org. 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 (accessed Oct. 10, 2023).

[39] Shah, S. S. H. sql injection dataset, Version 5. Retrieved June 10, 2023 from https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset.

[40] I. Crespo and A. Campazas, “SQL Injection Attack Netflow,” Zenodo, Jul. 26, 2022. https://zenodo.org/record/6907252 (accessed Jun. 27, 2023).

[41] B. Nagpal, N. Chauhan, and N. Singh, “A Survey on the Detection of SQL Injection Attacks and Their Countermeasures,” Journal of Information Processing Systems, vol. 13, no. 4, p. 691, Aug. 2017, doi: https://doi.org/10.3745/jips.03.0024.

[42] F. Moldovan, P. Sătmărean, and C. Oprişa, “An Analysis of HTTP Attacks on Home IoT Devices,” presented at the 2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), May 2020, pp. 1–6. doi: https://doi.org/10.1109/AQTR49680.2020.9129980.

[43] M. Wang, C. Jung, A. Ahad, and Y. Kwon, “Spinner: Automated Dynamic Command Subsystem Perturbation,” presented at the CCS ’21: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, Nov. 2021, pp. 1839–1860. doi: https://doi.org/10.1145/3460120.3484577.

[44] I. S. Crespo-Martínez, A. Campazas-Vega, Á. M. Guerrero-Higueras, V. Riego-DelCastillo, C. Álvarez-Aparicio, and C. Fernández-Llamas, “SQL injection attack detection in network flow data,” Computers & Security, vol. 127, p. 103093, Apr. 2023, doi: https://doi.org/10.1016/j.cose.2023.103093.

[45] I. Ghozali, M. F. Asy’ari, S. Triarjo, H. M. Ramadhani, H. Studiawan, and A. M. Shiddiqi, “A Novel SQL Injection Detection Using Bi-LSTM and TF-IDF,” presented at the 2022 7th International Conference on Information and Network Technologies (ICINT), IEEE, May 2022, pp. 1–8. doi: https://doi.org/10.1109/icint55083.2022.00010.

[46] E. M. Dharma, F. L. Gaol, H. L. H. S. Warnars, and B. Soewito, “THE ACCURACY COMPARISON AMONG WORD2VEC, GLOVE, AND FASTTEXT TOWARDS CONVOLUTION NEURAL NETWORK (CNN) TEXT CLASSIFICATION,” Journal of Theoretical and Applied Information Technology, vol. 100, no. 2, pp. 349–359, Jan. 2022.

[47] P. Ferreira, D. C. Le, and N. Zincir-Heywood, “Exploring Feature Normalization and Temporal Information for Machine Learning Based Insider Threat Detection,” in IEEE Xplore, Oct. 2019, pp. 1–7. doi: https://doi.org/10.23919/CNSM46954.2019.9012708.

[48] I. Shhadat, B. Bataineh, A. Hayajneh, and Z. A. Al-Sharif, “The Use of Machine Learning Techniques to Advance the Detection and Classification of Unknown Malware,” Procedia Computer Science, vol. 170, pp. 917–922, 2020, doi: https://doi.org/10.1016/j.procs.2020.03.110.

[49] A. A. Ismail and M. Yusoff, “An Efficient Hybrid LSTM-CNN and CNN-LSTM with GloVe for Text Multi-class Sentiment Classification in Gender Violence,” International Journal of Advanced Computer Science and Applications, vol. 13, no. 9, pp. 853–863, 2022, doi: https://doi.org/10.14569/ijacsa.2022.0130999.

[50] X. She and D. Zhang, “Text Classification Based on Hybrid CNN-LSTM Hybrid Model,” in 2018 11th International Symposium on Computational Intelligence

and Design (ISCID), IEEE, Dec. 2018, pp. 185–189. doi: https://doi.org/10.1109/ISCID.2018.10144.

[51] K. A. Torkura, M. I. H. Sukmana, A. V. D. M. Kayem, F. Cheng, and C. Meinel, “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), 2018, pp. 932–939. doi: https://doi.org/10.1109/BDCloud.2018.00137.

[52] H. Howland, “CVSS: Ubiquitous and Broken,” Digital Threats: Research and Practice, vol. 4, no. 1, pp. 1–12, Feb. 2022, doi: https://doi.org/10.1145/3491263.

[53] O. Suciu, C. Nelson, Z. Lyu, T. Bao, and T. Dumitras, “Expected Exploitability: Predicting the Development of Functional Vulnerability Exploits,” Feb. 2021.

[54] J. Williams, “OWASP Risk Rating Methodology,” owasp.org. https://owasp.org/www-community/OWASP_Risk_Rating_Methodology (accessed Jul. 17, 2023).

[55] P. Yermalovich and M. Mejri, “Risk Forecasting Automation on the Basis of MEHARI,” in ISSA 2020: Information and Cyber Security, Dec. 2020, pp. 34–49.

[56] B. Jung, Y. Li, and T. Bechor, “CAVP: A context-aware vulnerability prioritization model,” Computers & Security, vol. 116, p. 102639, May 2022, doi: https://doi.org/10.1016/j.cose.2022.102639.

[57] “Expected Exploitability,” www.exploitability.app. https://www.exploitability.app/faq (accessed Jul. 21, 2023).

[58] J. Jacobs, S. Romanosky, O. Suciu, B. Edwards, and A. Sarabi, “Enhancing Vulnerability Prioritization: Data-Driven Exploit Predictions with Community-Driven Insights,” in 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, Jul. 2023, pp. 194–206.

[59] Zach, “How to Normalize Data Between 0 and 100,” Statology, Nov. 30, 2020. https://www.statology.org/normalize-data-between-0-and-100/ (accessed Aug. 05, 2023).

[60] R. Ando and H. Itoh, “Characterizing Combatants of State-Sponsored APT in Digital Warfare by Reported Blocklist Database,” IJCSNS International Journal of Computer Science and Network Security, vol. 22, no. 3, pp. 541–546, Mar. 2022, doi: https://doi.org/10.22937/IJCSNS.2022.22.3.69.

[61] A. Ahmed, T. Ara, A. Kumar, M. Khanam, and M. Lutf, “Threat Intelligence Sharing: A Survey,” JASC: Journal of Applied Science and Computations, vol. 5, no. 11, pp. 1811–1815, Nov. 2018.

[62] M. Bada and I. Pete, “An exploration of the cybercrime ecosystem around Shodan,” in 2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Dec. 2020, pp. 1–8. doi: https://doi.org/10.1109/iotsms52051.2020.9340224.

[63] J. Hao and T. K. Ho, “Machine Learning Made Easy: A Review of Scikit-learn Package in Python Programming Language,” Journal of Educational and Behavioral Statistics, vol. 44, no. 3, p. 107699861983224, Feb. 2019, doi: https://doi.org/10.3102/1076998619832248.

[64] D. Zhang et al., “iBLP: An XGBoost-Based Predictor for Identifying Bioluminescent Proteins,” Developing and Applying Machine Learning-Based Methods in Special Function Protein Identification, vol. 2021, pp. 1–15, Jan. 2021, doi: https://doi.org/10.1155/2021/6664362.

[65] “Requests: HTTP for HumansTM — Requests 2.25.1 documentation,” docs.python-requests.org. https://docs.python-requests.org/en/latest/

[66] Xinghang Lv et al., “A Mitmproxy-based Dynamic Vulnerability Detection System For Android Applications,” in 2022 18th International Conference on Mobility, Sensing and Networking (MSN), IEEE, Dec. 2022, pp. 408–416. doi: https://doi.org/10.1109/msn57253.2022.00072.

[67] T. Lee, V. P. Singh, and K. Hwa Cho , “Tensorflow and Keras Programming for Deep Learning,” in Deep Learning for Hydrometeorology and Environmental Science, 2021, pp. 151–162.

[68] Kurti K. (2020), “Advanced-SQL-Injection-Cheatsheet” [GitHub]. https://github.com/kleiton0x00/Advanced-SQL-Injection-Cheatsheet/blob/main/MySQL-Bypass-Error/README.md

[69] “SQL error-based injection payloads - Documentation,” documentation.help. https://documentation.help/InfoSec-cn/cb9b96db-6970-4587-a204-7016f64e0ca4.htm (accessed Sep. 01, 2023).

9. Appendix

9.1 Dataset Count

Name	Source	Category	Count
Github	[34][35][36]	Authentication Bypass	247
SQLMap	[37]	Classic SQLi	15,474
SQLMap	[37]	Blind SQLi	16,039
OWASP, Kaggle, SQLMAP	[37][38][39]	Remote Code Execution	54
Kaggle	[39]	Benign (Normal Text + SQL Operations)	37,709
Kaggle	[39]	Denial of Service	320
Zenodo	[40]	Netflow (Malicious and Benign)	457,233

Table 17 - Dataset Compilation Sources

9.2 Classifier Evaluation

	Class	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	FPR
SVM	0	99.94%	95.60%	87.88%	91.58%	0.01436
	1	99.61%	99.81%	98.47%	99.14%	0.05575
	2	100.00%	100.00%	100.00%	100.00%	0.0000
	3	99.97%	99.97%	99.90%	99.94%	0.00920

	4	99.98%	100.00%	76.19%	86.49%	0.0000
	5	99.52%	99.22%	99.90%	99.56%	0.91800
RF	0	99.95%	95.70%	89.90%	92.71%	0.01437
	1	99.92%	99.74%	99.94%	99.84%	0.07898
	2	99.99%	100.00%	98.44%	99.21%	0.0000
	3	99.99%	100.00%	99.95%	99.98%	0.0000
	4	99.98%	100.00%	99.95%	99.98%	0.0000
	5	99.86%	99.86%	99.88%	99.87%	0.16337
XGBoost	0	99.94%	96.67%	87.88%	92.06%	0.01078
	1	99.85%	99.56%	99.77%	99.67%	0.13010
	2	99.99%	100.00%	98.44%	99.21%	0.0000
	3	99.99%	100.00%	99.94%	99.97%	0.0100
	4	99.98%	100.00%	71.43%	83.33%	0.0000
	5	99.76%	98.75%	99.81%	99.78%	0.28784
NB	0	99.65%	96.34%	79.80%	87.29%	0.01078
	1	97.98%	95.99%	95.18%	95.59%	1.18483
	2	99.54%	94.73%	42.19%	58.38%	0.01079
	3	97.99%	95.03%	95.98%	95.50%	1.43002
	4	99.92%	100.00%	61.90%	76.47%	0.00000
	5	98.10%	97.48%	99.03%	98.25%	3.00296
	0	99.92%	96.34%	79.80%	87.29%	0.01078

CNN	1	99.67%	98.72%	99.84%	99.28%	0.38565
	2	99.72%	94.74%	42.19%	58.38%	0.01079
	3	99.99%	99.98%	99.97%	99.98%	0.00460
	4	99.97%	100.00%	61.90%	76.47%	0.0000
	5	99.88%	99.81%	99.97%	99.89%	0.2256
LSTM	0	99.95%	92.93%	92.93%	92.93%	0.02514
	1	99.96%	99.91%	99.94%	99.92%	0.02788
	2	100.00%	100.00%	100.00%	100.00%	0.00000
	3	99.99%	99.98%	99.98%	99.98%	0.00460
	4	99.99%	94.74%	85.71%	90.00%	0.00358
	5	99.91%	99.91%	99.91%	99.91%	0.10113
CNN + LSTM	0	99.93%	87.74%	93.94%	90.73%	0.04670
	1	99.96%	99.89%	99.95%	99.92%	0.03252
	2	100.00%	100.00%	100.00%	100.00%	0.00000
	3	100.00%	99.98%	100.00%	99.99%	0.00460
	4	99.98%	100.00%	76.19%	86.49%	0.00000
	5	99.90%	99.93%	99.88%	99.90%	0.08558

Table 18 - ML Model Performance: Individual Classes

	Class	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1_Score (%)**	**FPR**
SVM	malicious	94.48%	90.16%	99.86%	94.76%	0.1091
	benign	94.48%	99.85%	89.09%	94.16%	0.0014

RF	malicious	85.92%	78.86%	98.19%	87.47%	0.2694
	benign	85.92%	97.60%	73.64%	83.94%	0.0181
XGBoost	malicious	74.33%	66.12%	99.87%	79.56%	0.5125
	benign	74.33%	99.74%	48.75%	65.49%	0.0013
NB	malicious	94.87%	92.91%	97.16%	94.99%	0.0742
	benign	94.87%	97.02%	92.58%	94.75%	0.0284
CNN	malicious	94.89%	90.81%	99.88%	95.13%	0.1012
	benign	94.89%	99.87%	89.88%	94.61%	0.0012
LSTM	malicious	61.62%	99.68%	23.36%	37.85%	0.0007
	benign	61.62%	56.56%	99.93%	72.23%	0.7664
CNN + LSTM	malicious	97.63%	96.00%	99.40%	97.67%	0.0414
	benign	97.63%	99.38%	95.86%	97.59%	0.0060

Table 19 - NetFlow ML Model Performance: Individual Classes

9.3 Risk R34P3R – Risk Rating Comparisons

Authentication Bypass				
CVE	**CVSS Severity**	**EPSS**	**EE**	**RISK R34P3R**
CVE-2023-36284	7.5	0.000760000	0.0109	Medium
CVE-2023-33178	6.5	0.000650000	1.0	Medium
CVE-2023-26034	8.8	0.001200000	1.0	High

CVE-2023-22319	9.8	0.000760000	0.0004	Medium
CVE-2023-1016	7.2	0.000500000	0.0549	Medium
CVE-2022-36669	9.8	0.000910000	1.0	High
CVE-2022-34909	9.1	0.000760000	1.0	High
CVE-2022-29009	9.8	0.200330000	0.0002	High
CVE-2022-29007	9.8	0.200330000	0.1153	High
CVE-2022-29006	9.8	0.200330000	0.6759	Critical
CVE-2021-45814	9.8	0.006610000	1.0	High
CVE-2021-45334	9.8	0.005680000	1.0	High
CVE-2021-44966	9.8	0.001710000	1.0	High
CVE-2021-44655	9.8	0.006610000	1.0	High
CVE-2021-44653	9.8	0.006610000	1.0	High
CVE-2021-44088	9.8	0.001750000	1.0	High
CVE-2021-42665	9.8	0.002690000	1.0	High
CVE-2021-42580	9.8	0.020190000	0.0007	Medium
CVE-2021-42169	9.8	0.008750000	4.7937	Medium
CVE-2021-41511	9.8	0.004430000	1.0	High
CVE-2021-41433	9.8	0.000760000	1.653	Medium
CVE-2021-38167	9.8	0.001380000	1.0	High
CVE-2021-36624	9.8	0.002960000	1.0	High
CVE-2021-34166	9.8	0.001340000	0.0001	Medium
CVE-2021-34165	9.8	0.001340000	2.8877	Medium
CVE-2021-33578	9.8	0.001380000	1.8821	Medium
CVE-2021-3278	9.8	0.010070000	1.0	High
CVE-2021-27130	9.8	0.001630000	1.0	High

CVE-2021-26201	9.8	0.002070000	0.9175	High
CVE-2021-26200	9.8	0.002070000	0.2709	Medium
CVE-2020-9465	9.8	0.001640000	1.0	High
CVE-2020-8656	9.8	0.069790000	1.0	Critical
CVE-2020-8427	9.8	0.002390000	0.0004	Medium
CVE-2020-5511	8.8	0.000920000	1.0	High
CVE-2020-5257	8.1	0.000630000	1.0	High
CVE-2020-35427	9.8	0.004730000	0.0003	Medium
CVE-2020-35378	9.8	0.002570000	0.9995	High
CVE-2020-29282	9.8	0.005690000	0.8118	High
CVE-2020-29214	9.8	0.002340000	7.853	Medium
CVE-2020-28172	9.8	0.002970000	0.1489	Medium
CVE-2020-28133	9.8	0.002570000	1.0	High
CVE-2020-28074	9.8	0.002160000	1.0	High
CVE-2020-28073	9.8	0.004540000	1.0	High
CVE-2020-25952	9.8	0.017670000	1.0	High
CVE-2020-25889	9.8	0.007180000	0.0102	Medium
CVE-2020-25762	9.1	0.011190000	1.0	High
CVE-2020-25273	9.8	0.002000000	1.0	High
CVE-2020-25132	9.8	0.002020000	1.0	High
CVE-2020-25130	6.5	0.000760000	1.0	Medium
CVE-2020-24208	9.8	0.002540000	1.0	High
CVE-2020-24193	9.8	0.002230000	1.0	High
CVE-2020-23763	9.8	0.001330000	1.0	High
CVE-2020-15849	7.2	0.000980000	0.0003	Medium

CVE-2020-14972	9.8	0.007040000	0.9281	High
CVE-2020-14068	9.8	0.001510000	0.0011	Medium
CVE-2020-14054	9.8	0.001650000	1.0	High
CVE-2020-13873	9.8	0.017570000	0.9838	High
CVE-2020-11545	9.8	0.002100000	0.5767	Medium
Blind SQLi CVEs				
CVE-2023-34487	9.8	0.000760000	0.0004	Medium
CVE-2023-33280	9.8	0.000760000	0.9195	High
CVE-2023-33279	9.8	0.000760000	0.415	Medium
CVE-2023-33278	9.8	0.000760000	0.9791	High
CVE-2023-32308	9.8	0.000760000	0.8889	High
CVE-2023-32306	9.8	0.000910000	0.9977	High
CVE-2023-3197	9.8	0.000800000	2.1466	Medium
CVE-2023-3077	9.8	0.003730000	0.0045	Medium
CVE-2023-29842	9.8	0.000700000	1.0	High
CVE-2023-28883	9.8	0.000760000	1.0	High
CVE-2023-26034	8.8	0.001200000	1.0	High
CVE-2023-23315	9.8	0.000760000	1.0	High
CVE-2023-2080	9.8	0.000760000	1.5142	Medium
CVE-2023-0875	8.8	0.000500000	1.0	High
CVE-2023-0765	8.8	0.000500000	1.0	High
CVE-2022-34972	9.8	0.001410000	1.0	High
CVE-2022-3141	8.8	0.000740000	0.8707	High
CVE-2022-31296	9.8	0.001320000	0.0024	Medium
CVE-2022-30493	9.8	0.001190000	5.9116	Medium

CVE-2022-30004	9.8	0.000720000	1.0	High
CVE-2022-29689	7.2	0.000720000	9.0572	Medium
CVE-2022-29688	7.2	0.000720000	9.0572	Medium
CVE-2022-29687	7.2	0.000720000	6.5904	Medium
CVE-2022-29686	7.2	0.000720000	8.384	Medium
CVE-2022-29685	8.8	0.000720000	6.5904	Medium
CVE-2022-29684	7.2	0.000720000	4.943	Medium
CVE-2022-29683	7.2	0.000720000	4.943	Medium
CVE-2022-29682	7.2	0.000720000	0.0009	Medium
CVE-2022-29681	7.2	0.000720000	0.0001	Medium
CVE-2022-29680	7.2	0.000720000	6.5904	Medium
CVE-2022-29661	7.2	0.000720000	1.1217	Medium
CVE-2022-29305	8.1	0.001390000	1.5478	Medium
CVE-2022-28111	9.8	0.002590000	0.0002	Medium
CVE-2022-28105	9.8	0.001390000	0.0168	Medium
CVE-2022-27366	7.2	0.000720000	2.326	Medium
CVE-2022-27175	9.8	0.001200000	1.0	High
CVE-2022-27104	9.8	0.001500000	1.0896	Medium
CVE-2022-26959	9.8	0.000860000	0.2207	Medium
CVE-2022-26887	9.8	0.001200000	1.0	High
CVE-2022-26836	9.8	0.001200000	1.0	High
CVE-2022-26667	9.8	0.001200000	1.0	High
CVE-2022-26666	9.8	0.001200000	1.0	High
CVE-2022-26632	9.8	0.001670000	0.005	Medium
CVE-2022-26631	9.8	0.001520000	7.5135	Medium

CVE-2022-26514	9.8	0.001200000	1.0	High
CVE-2022-26349	9.8	0.001200000	1.0	High
CVE-2022-26338	9.8	0.001200000	1.0	High
CVE-2022-26069	9.8	0.001200000	1.0	High
CVE-2022-26065	9.8	0.001200000	1.0	High
CVE-2022-26059	9.8	0.001200000	1.0	High
CVE-2022-26013	9.8	0.001200000	1.0	High
CVE-2022-25980	9.8	0.001200000	1.0	High
CVE-2022-25880	9.8	0.001200000	1.0	High
CVE-2022-24956	6.5	0.000690000	1.0	Medium
CVE-2022-24707	8.8	0.002850000	1.0	High
CVE-2022-24691	7.1	0.000610000	1.0	High
CVE-2022-24690	8.2	0.001540000	1.0	High
CVE-2022-24226	7.5	0.001680000	1.0	High
CVE-2022-23387	7.5	0.001320000	0.002	Medium
CVE-2022-1378	9.8	0.001200000	1.0	High
CVE-2022-1377	9.8	0.001200000	1.0	High
CVE-2022-1376	9.8	0.001200000	1.0	High
CVE-2022-1375	9.8	0.001200000	1.0	High
CVE-2022-1374	9.8	0.001200000	1.0	High
CVE-2022-1372	9.8	0.001200000	1.0	High
CVE-2022-1371	9.8	0.001200000	1.0	High
CVE-2022-1370	9.8	0.001200000	1.0	High
CVE-2022-1369	9.8	0.001200000	1.0	High
CVE-2022-1367	9.8	0.001200000	1.0	High

CVE-2022-1366	9.8	0.001200000	1.0	High
CVE-2022-1258	7.2	0.000830000	0.9971	High
CVE-2022-1013	9.8	0.011440000	0.1233	Medium
CVE-2022-0923	9.8	0.001200000	1.0	High
CVE-2022-0842	4.9	0.000740000	1.0	Medium
CVE-2022-0439	8.8	0.000620000	0.0002	Medium
CVE-2022-0349	9.8	0.011250000	0.0015	Medium
CVE-2021-45821	8.8	0.003450000	0.0002	Medium
CVE-2021-44915	7.2	0.000720000	8.643	Medium
CVE-2021-44249	9.8	0.003190000	0.0048	Medium
CVE-2021-43969	6.5	0.000670000	1.0	Medium
CVE-2021-43789	9.8	0.001730000	1.0	High
CVE-2021-43701	6.5	0.004270000	1.0	Medium
CVE-2021-41932	8.8	0.000720000	3.731	Medium
CVE-2021-41920	7.5	0.002080000	1.0	High
CVE-2021-41651	7.5	0.001290000	1.0	High
CVE-2021-41647	9.1	0.001870000	1.0	High
CVE-2021-41609	9.8	0.001750000	1.0477	Medium
CVE-2021-40578	7.2	0.001600000	0.0072	Medium
CVE-2021-38706	8.8	0.000870000	1.0	High
CVE-2021-3860	8.8	0.000870000	0.0085	Medium
CVE-2021-38393	9.8	0.002720000	0.0158	Medium
CVE-2021-38391	9.8	0.002720000	0.4797	Medium
CVE-2021-38390	9.8	0.002720000	0.0158	Medium
CVE-2021-37808	5.9	0.003980000	0.3562	Low

CVE-2021-37806	5.9	0.003840000	0.9824	Medium
CVE-2021-37749	9.8	0.001630000	1.0	High
CVE-2021-3604	9.8	0.001400000	1.0	High
CVE-2021-35487	6.5	0.000670000	0.0432	Low
CVE-2021-35212	8.8	0.002750000	0.0971	Medium
CVE-2021-32983	9.8	0.002720000	0.13	Medium
CVE-2021-32582	7.5	0.001360000	0.4566	Medium
CVE-2021-31867	7.5	0.001840000	0.0009	Medium
CVE-2021-30486	8.8	0.000880000	0.1048	Medium
CVE-2021-30117	8.8	0.000870000	1.0	High
CVE-2021-28419	7.2	0.172360000	0.0047	Medium
CVE-2021-28022	7.5	0.001640000	1.0	High
CVE-2021-27320	7.5	0.203330000	0.6654	High
CVE-2021-27319	7.5	0.159620000	0.6674	High
CVE-2021-27316	7.5	0.159620000	0.6654	High
CVE-2021-27315	7.5	0.159620000	0.9222	High
CVE-2021-25899	7.5	0.545550000	1.0	Critical
CVE-2021-25784	7.2	0.000880000	1.5152	Medium
CVE-2021-25783	7.2	0.000880000	5.51	Medium
CVE-2021-24747	7.2	0.000890000	2.6003	Medium
CVE-2021-24360	6.5	0.000760000	1.5807	Low
CVE-2021-24345	6.6	0.001140000	0.9962	High
CVE-2021-24295	7.5	0.001290000	0.0018	Medium
CVE-2021-24200	6.5	0.001090000	8.1391	Low
CVE-2021-24199	6.5	0.001090000	4.463	Low

CVE-2021-23837	6.5	0.003590000	0.0081	Low
CVE-2021-21024	9.1	0.000750000	0.3203	Medium
CVE-2020-5920	4.3	0.000540000	1.0	Medium
CVE-2020-3973	8.8	0.001010000	1.0	High
CVE-2020-36112	9.8	0.480490000	1.0	Critical
CVE-2020-36003	7.5	0.001320000	0.0007	Medium
CVE-2020-29015	9.8	0.001360000	1.0	High
CVE-2020-28860	8.8	0.006520000	1.0	High
CVE-2020-26248	8.2	0.014880000	0.0537	Medium
CVE-2020-25362	7.5	0.006480000	0.0568	Medium
CVE-2020-24862	7.5	0.006480000	0.005	Medium
CVE-2020-24569	4.3	0.000540000	1.0	Medium
CVE-2020-24568	6.5	0.000650000	1.0	Medium
CVE-2020-23630	8.8	0.001770000	0.771	High
CVE-2020-21726	9.8	0.001720000	0.0393	Medium
CVE-2020-21725	9.8	0.001720000	0.112	Medium
Classic SQLi CVEs				
CVE-2022-43859	4.3	0.000760000	0.441	Low
CVE-2022-2491	8.8	0.000760000	0.0165	Medium
CVE-2022-24707	8.8	0.000440000	1.0	High
CVE-2022-2086	8.8	0.000800000	1.0	High
CVE-2021-44593	8.1	0.002850000	1.0	High
CVE-2021-41609	9.8	0.000800000	1.0477	Medium
CVE-2021-35458	9.8	0.005520000	0.82	High
CVE-2021-30117	8.8	0.001870000	1.0	High

CVE-2021-24186	6.5	0.001750000	1.162	Low
CVE-2021-24183	6.5	0.001600000	9.1161	Low
CVE-2021-24182	6.5	0.001380000	9.1161	Low
CVE-2020-7759	7.2	0.010760000	1.0	High
CVE-2020-36003	7.5	0.000870000	0.0007	Medium
CVE-2020-22807	9.8	0.000890000	0.8776	High
CVE-2020-15226	4.3	0.000850000	1.0	Medium
CVE-2020-10505	9.8	0.000850000	1.0	High
CVE-2023-34735	9.8	0.000850000	0.0202	Medium
CVE-2023-32308	9.8	0.001040000	0.8889	High
CVE-2021-41647	9.1	0.480490000	1.0	Critical
CVE-2021-40578	7.2	0.001320000	0.0072	Medium
CVE-2021-36722	9.8	0.006480000	0.0117	Medium
CVE-2021-24747	7.2	0.002450000	2.6003	Medium
CVE-2020-36112	9.8	0.000580000	1.0	High
CVE-2020-25362	7.5	0.001260000	0.0568	Medium
CVE-2020-15072	8.8	0.001520000	0.9062	High
Denial-of-Service SQLi CVEs				
CVE-2023-27649	7.5	0.000560000	7.7572	Medium
CVE-2023-2760	7.6	0.000480000	1.1609	Medium
CVE-2023-26033	9.1	0.000480000	1.0	High
CVE-2022-41271	9.4	0.000750000	9.5293	Medium
CVE-2021-36299	8.1	0.000820000	0.0083	Medium
CVE-2021-3119	7.5	0.001090000	2.3226	Medium
CVE-2020-8158	9.8	0.002820000	4.5645	High

CVE-2020-27207	7.5	0.001730000	0.0001	Medium
CVE-2020-10184	7.5	0.001150000	0.0002	Medium
Remote Code Execution SQLi CVEs				
CVE-2023-34362	9.8	0.936790000	1.0	Critical
CVE-2023-32530	8.8	0.003400000	0.2076	Medium
CVE-2023-32529	8.8	0.003400000	0.2076	Medium
CVE-2023-31702	7.2	0.001690000	1.0	High
CVE-2023-30625	8.8	0.009160000	0.6788	High
CVE-2023-30246	9.8	0.001070000	0.009	Medium
CVE-2023-30245	9.8	0.001070000	0.0096	Medium
CVE-2023-29809	9.8	0.024830000	1.0	High
CVE-2023-27709	7.2	0.000610000	1.0	High
CVE-2023-27707	7.2	0.000610000	1.0	High
CVE-2023-26922	9.8	0.000910000	1.0	High
CVE-2023-26876	8.8	0.012790000	1.0	High
CVE-2023-26750	9.8	0.001500000	1.0	High
CVE-2023-26034	8.8	0.001200000	1.0	High
CVE-2022-46764	9.8	0.002120000	1.0	High
CVE-2022-43775	9.8	0.001000000	1.0	High
CVE-2022-43774	9.8	0.001000000	1.0	High
CVE-2022-39179	7.2	0.002000000	0.0172	Medium
CVE-2022-36961	8.8	0.003550000	0.0195	Medium
CVE-2022-29807	9.8	0.001720000	0.0094	Medium
CVE-2022-24688	8.8	0.000780000	0.8629	High
CVE-2022-22794	9.8	0.001270000	9.8792	Medium

CVE-2022-21647	9.8	0.085740000	0.0056	Medium
CVE-2022-1531	9.8	0.001860000	5.4221	Medium
CVE-2021-45821	8.8	0.003450000	0.0002	Medium
CVE-2021-44593	8.1	0.005520000	1.0	High
CVE-2021-43630	8.8	0.001300000	9.9097	Medium
CVE-2021-43035	9.8	0.007390000	7.5848	Medium
CVE-2021-42670	9.8	0.009720000	1.0	High
CVE-2021-42668	9.8	0.008810000	1.0	High
CVE-2021-42667	9.8	0.021190000	1.0	High
CVE-2021-42666	8.8	0.021250000	1.0	High
CVE-2021-42258	9.8	0.974660000	0.2072	High
CVE-2021-41765	9.8	0.064180000	1.0	High
CVE-2021-41662	9.8	0.001420000	3.1926	Medium
CVE-2021-4088	7.2	0.000930000	0.9063	High
CVE-2021-40843	7.3	0.000460000	7.2507	Medium
CVE-2021-38393	9.8	0.002720000	0.0158	Medium
CVE-2021-38391	9.8	0.002720000	0.4797	Medium
CVE-2021-38390	9.8	0.002720000	0.0158	Medium
CVE-2021-37599	9.8	0.001900000	0.0066	Medium
CVE-2021-37358	9.8	0.001490000	0.0004	Medium
CVE-2021-33701	9.1	0.007420000	1.0	High
CVE-2021-32983	9.8	0.002720000	0.13	Medium
CVE-2021-3239	9.8	0.031840000	1.0	High
CVE-2021-30177	9.8	0.001490000	1.0	High
CVE-2021-27644	8.8	0.005500000	2.0558	Medium

CVE-2021-26822	9.8	0.020230000	0.0209	Medium
CVE-2021-26644	9.8	0.001930000	8.6033	Medium
CVE-2021-26636	9.6	0.001200000	0.0012	Medium
CVE-2020-9006	9.8	0.005970000	1.0	High
CVE-2020-36077	8.8	0.000710000	0.0002	Medium
CVE-2020-36074	8.8	0.000600000	0.0006	Medium
CVE-2020-36073	8.8	0.000600000	0.0083	Medium
CVE-2020-36072	8.8	0.000600000	0.0093	Medium
CVE-2020-36071	8.8	0.000600000	0.0002	Medium
CVE-2020-35701	8.8	0.004810000	0.0158	Medium
CVE-2020-29474	9.8	0.015030000	0.5927	Medium
CVE-2020-29472	9.8	0.015030000	0.0273	Medium
CVE-2020-29143	7.2	0.001540000	0.9838	High
CVE-2020-29140	7.2	0.002100000	1.0	High
CVE-2020-28070	9.8	0.004170000	1.0	High
CVE-2020-25006	9.8	0.002010000	1.0	High
CVE-2020-25005	9.8	0.001670000	1.0	High
CVE-2020-25004	9.8	0.001670000	0.6904	High
CVE-2020-23833	9.8	0.029920000	0.0005	Medium
CVE-2020-21400	7.2	0.000600000	0.0002	Medium
CVE-2020-20915	9.8	0.000910000	0.0013	Medium
CVE-2020-20914	9.8	0.000910000	0.005	Medium
CVE-2020-20913	9.8	0.000910000	0.0001	Medium
CVE-2020-20491	7.2	0.000600000	0.0107	Medium
CVE-2020-20413	9.8	0.000910000	8.3446	Medium

CVE-2020-19114	9.8	0.002040000	8.5677	Medium
CVE-2020-19112	9.8	0.002040000	8.7843	Medium
CVE-2020-19110	9.8	0.001680000	8.7843	Medium
CVE-2020-19109	9.8	0.002040000	8.7843	Medium
CVE-2020-19108	9.8	0.002040000	8.7843	Medium
CVE-2020-19107	9.8	0.002040000	8.7843	Medium
CVE-2020-18746	7.2	0.001190000	0.0002	Medium
CVE-2020-18717	9.8	0.001490000	0.9291	High
CVE-2020-18544	9.8	0.001490000	0.0043	Medium
CVE-2020-18215	8.8	0.001370000	0.0885	Medium
CVE-2020-18020	9.8	0.001500000	0.0041	Medium
CVE-2020-15849	7.2	0.000980000	0.0003	Medium
CVE-2020-15394	9.8	0.004360000	0.133	Medium
CVE-2020-14972	9.8	0.007040000	0.9281	High
CVE-2020-14497	9.8	0.008410000	1.0	High
CVE-2020-13877	9.8	0.001580000	0.42	Medium
CVE-2020-13873	9.8	0.017570000	0.9838	High
CVE-2020-12271	9.8	0.011000000	1.0	High

Table 20 - Risk R34P3R – Risk Rating Comparisons

SQLi Class	**CVSS Severity**	**EPSS**	**EE**	**RISK R34P3R**
Authentication Bypass	Critical	0.015158	0.663932	High
Blind SQLi	High	0.013951	0.543993	Medium
Denial-of-Service SQLi	High	0.001098	0.112075	Medium
Classic SQLi	High	0.021313	0.521872	Medium

Remote Code Execution SQLi	Critical	0.027397	0.418329	Medium

Table 21 - SQLi Average Class Risk Ratings

9.4 System Evaluation – Detection and Classification of SQLi Attack Classes

Figure 28 - Example 1: Classic SQLi Detection and Classification

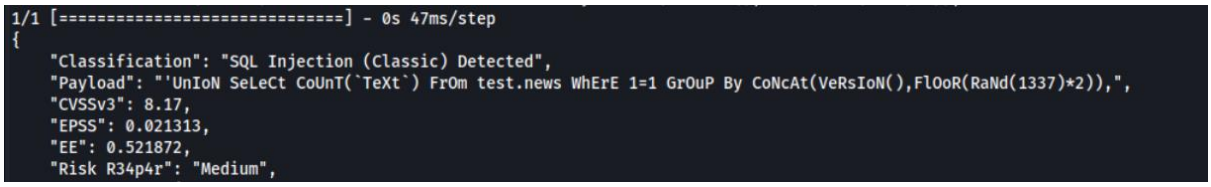


Figure 29 - Example 2: Classic SQLi Detection and Classification

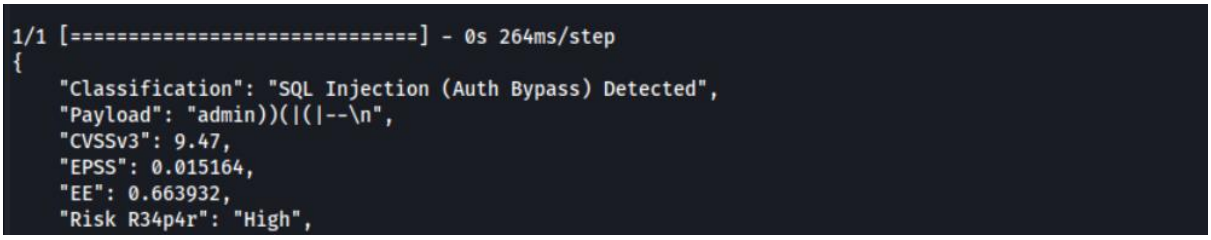


Figure 30 - Example 1: Authentication Bypass SQLi Detection and Classification

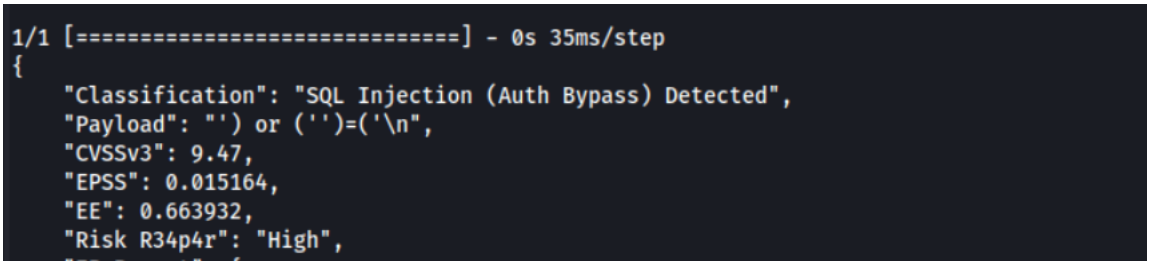


Figure 31 - Example 2: Authentication Bypass SQLi Detection and Classification

Figure 32 - Example 1: Remote Code Execution SQLi Detection and Classification

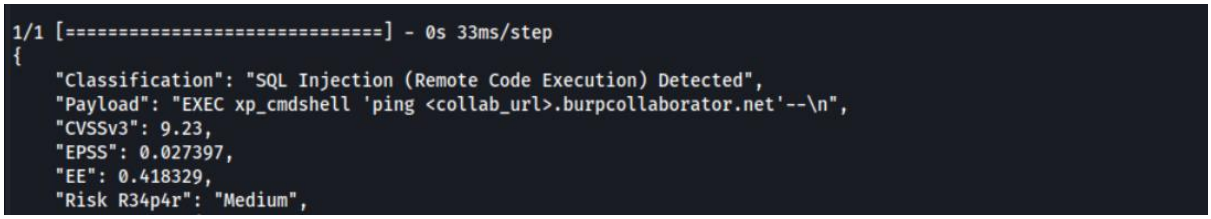


Figure 33 - Example 2: Remote Code Execution SQLi Detection and Classification

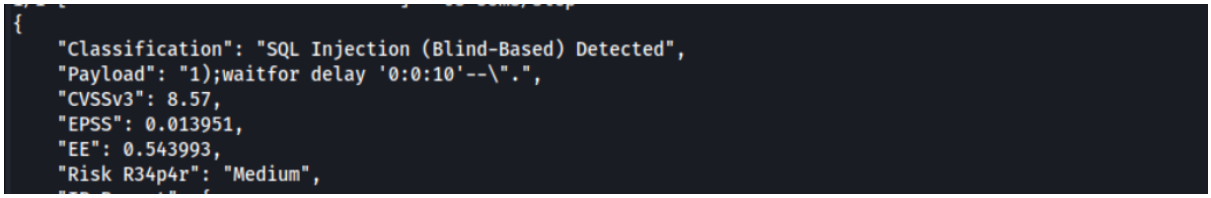


Figure 34 - Example 1: Blind SQLi Detection and Classification

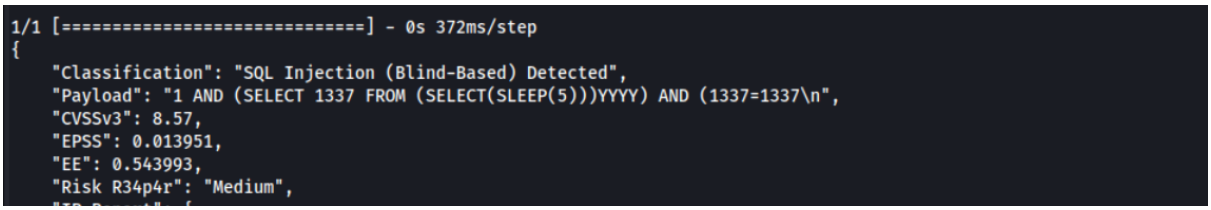


Figure 35 - Example 2: Blind SQLi Detection and Classification

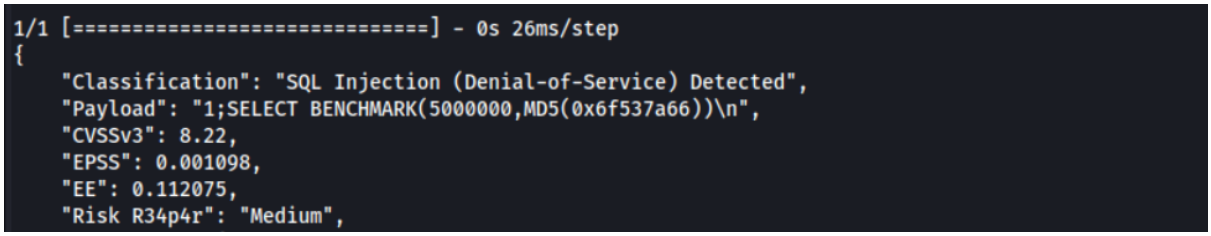


Figure 36 - Example 1: Denial-of-Service SQLi Detection and Classification

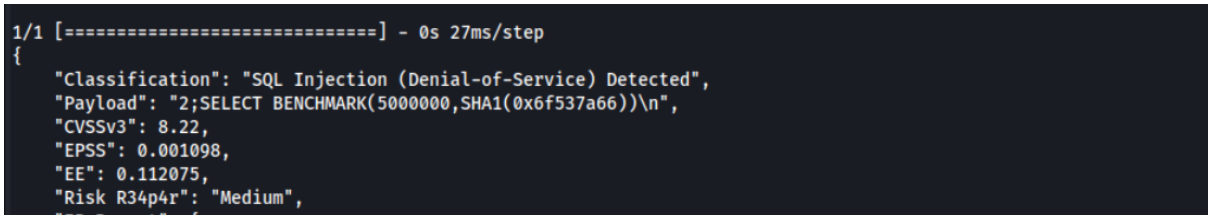
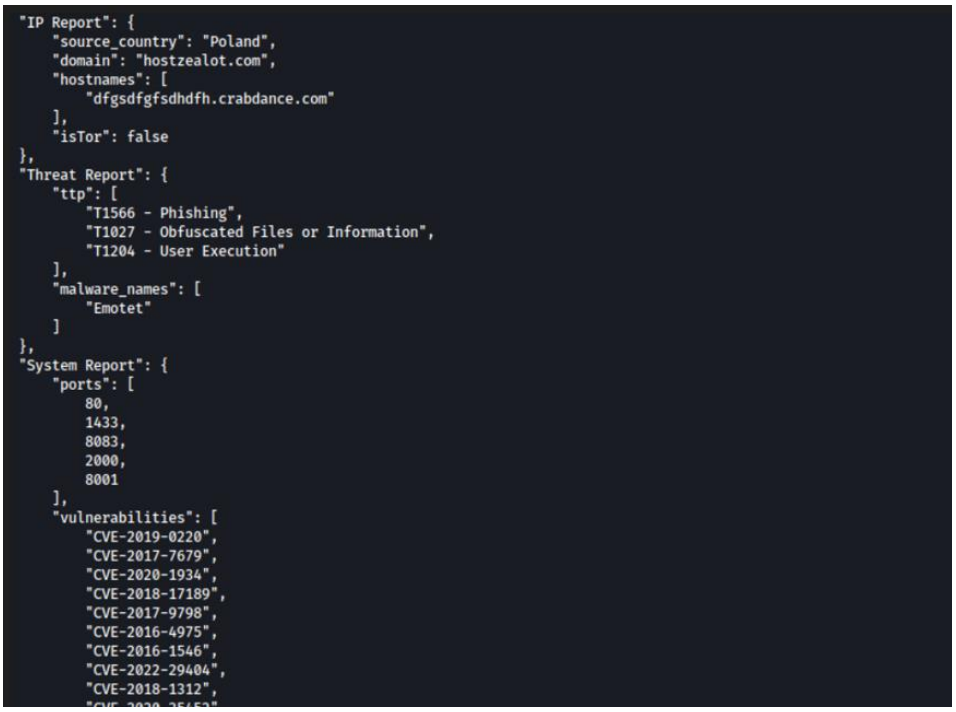


Figure 37 - Example 2: Denial-of-Service SQLi Detection and Classification

9.5 System Evaluation – Threat Modelling of SQLi Attack Classes



```
"IP Report": {
    "source_country": "Poland",
    "domain": "hostzealot.com",
    "hostnames": [
        "dfgsdfgfsdhdfh.crabdance.com"
    ],
    "isTor": false
},
"Threat Report": {
    "ttp": [
        "T1566 - Phishing",
        "T1027 - Obfuscated Files or Information",
        "T1204 - User Execution"
    ],
    "malware_names": [
        "Emotet"
    ]
},
"System Report": {
    "ports": [
        80,
        1433,
        8083,
        2000,
        8001
    ],
    "vulnerabilities": [
        "CVE-2019-0220",
        "CVE-2017-7679",
        "CVE-2020-1934",
        "CVE-2018-17189",
        "CVE-2017-9798",
        "CVE-2016-4975",
        "CVE-2016-1546",
        "CVE-2022-29404",
        "CVE-2018-1312",
```


Figure 38 - Threat Modelling Output for Detected SQLi Attack

9.6 System Evaluation – Complete Output

```
{
    "Classification": "SQL Injection (Classic) Detected",
    "Payload": "'UnIoN SeLeCt CoUnT(`TeXt`) FrOm test.news WhErE 1=1 GrOuP By CoNcAt(VeRsIoN(),FlOoR(RaNd(1337)*2)),",
    "CVSSv3": 8.17,
    "EPSS": 0.021313,
    "EE": 0.521872,
    "Risk R34p4r": "Medium",
    "IP Report": {
        "source_country": "Poland",
        "domain": "hostzealot.com",
        "hostnames": [
            "dfgsdfgfsdhdfh.crabdance.com"
        ],
        "isTor": false
    },
    "Threat Report": {
        "ttp": [
            "T1566 - Phishing",
            "T1027 - Obfuscated Files or Information",
            "T1204 - User Execution"
        ],
        "malware_names": [
            "Emotet"
        ]
    },
    "System Report": {
        "ports": [
            80,
            1433,
            8083,
            2000,
            8001
        ],
        "vulnerabilities": [
            "CVE-2019-0220",
            "CVE-2017-7679",
            "CVE-2020-1934",
            "CVE-2018-17189",
            "CVE-2017-9798",
            "CVE-2016-4975",
            "CVE-2016-1546",
            "CVE-2022-29404",
            "CVE-2018-1312",
            "CVE-2020-35452",
            "CVE-2018-1333",
            "CVE-2019-0211",
            "CVE-2018-11763",
            "CVE-2022-28330",
            "CVE-2017-15710",
            "CVE-2016-8612",
            "CVE-2019-0217",
            "CVE-2019-0196",
            "CVE-2022-22721",
            "CVE-2022-22720",
            "CVE-2019-10092",
            "CVE-2019-17567",
            "CVE-2017-15715",
            "CVE-2022-31813",
            "CVE-2019-10098",
            "CVE-2022-37436",
            "CVE-2016-5387",
            "CVE-2021-40438",
            "CVE-2022-23943",
            "CVE-2020-1927",
            "CVE-2018-17199",
            "CVE-2017-9788",
            "CVE-2022-22719",
            "CVE-2018-1301",
            "CVE-2018-1302",
            "CVE-2018-1303",
            "CVE-2017-3167",
            "CVE-2021-34798",
            "CVE-2023-25690",
            "CVE-2017-3169",
            "CVE-2020-11985",
            "CVE-2021-44790",
            "CVE-2016-4979",
            "CVE-2021-26690",
            "CVE-2021-26691",
            "CVE-2022-26377",
            "CVE-2022-30556",
            "CVE-2020-13938",
            "CVE-2006-20001",
            "CVE-2018-1283",
            "CVE-2019-10082",
            "CVE-2016-8740",
            "CVE-2016-8743",
            "CVE-2021-44224",
            "CVE-2021-39275",
            "CVE-2022-28615",
            "CVE-2022-28614",
            "CVE-2022-36760",
            "CVE-2021-33193"
        ]
```


Figure 39 - SQLR34P3R Complete Output

9.7 System Evaluation – Multisource (Network Traffic and Header Injection)

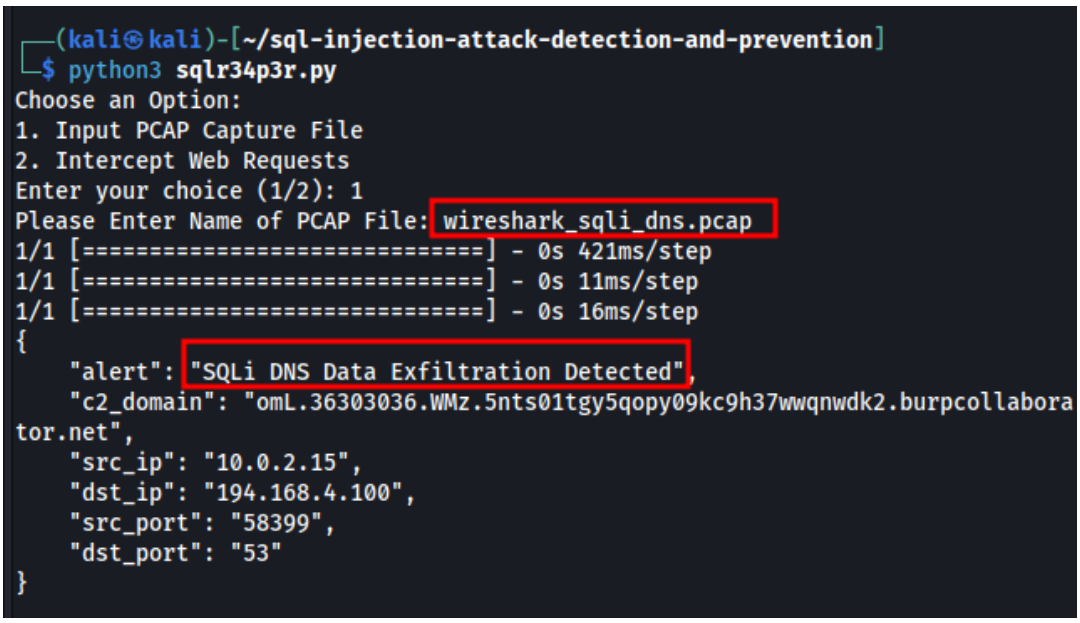


Figure 40 - SQLi DNS Data Exfiltration Detection

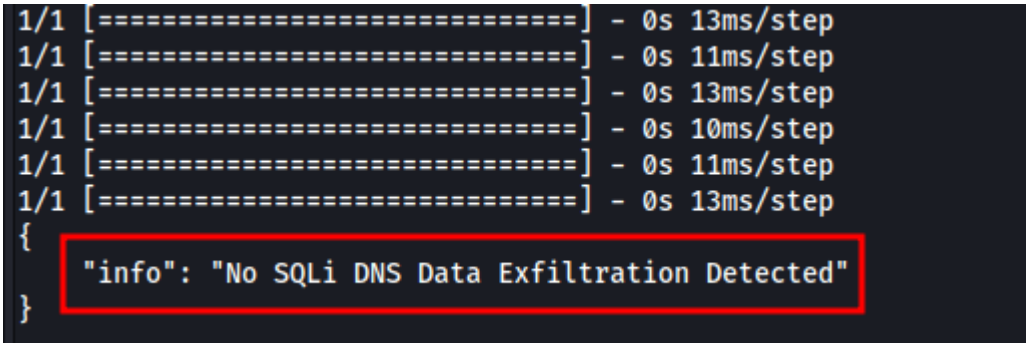


Figure 41 - No SQLi Exfiltration Detected

Figure 42 - User-Agent Header Injection

Figure 43 - Referer Header Injection

Figure 44 – X-Forwarded-For Header Injection

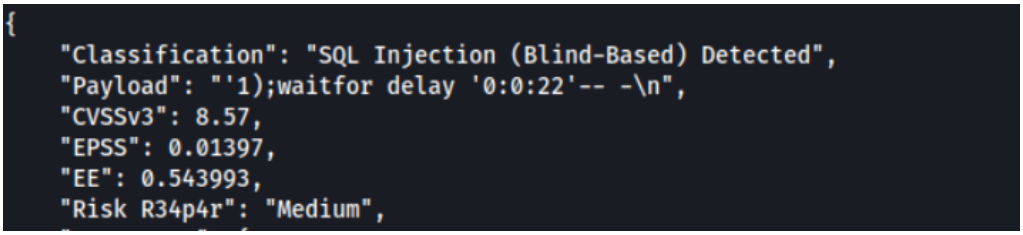


Figure 45 SQLR34P3R: User-Agent SQLi Detection

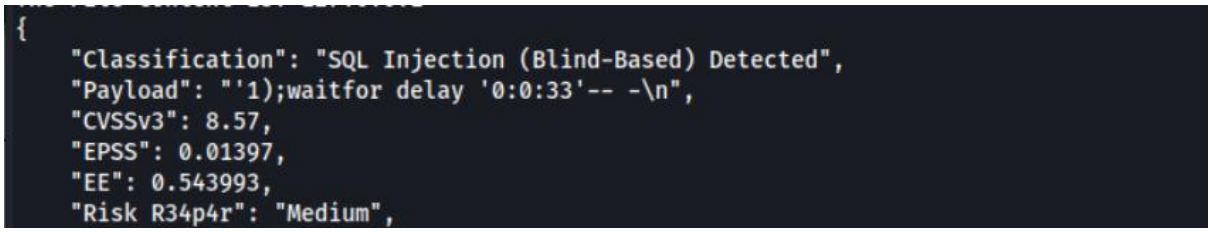


Figure 46 - SQLR34P3R Referer SQLi Detection

Figure 47 - SQLR34P3R: X-Forwarded-For SQLi Detection

9.8 System Evaluation – Prevention (Request Filtering through IP Blocking)

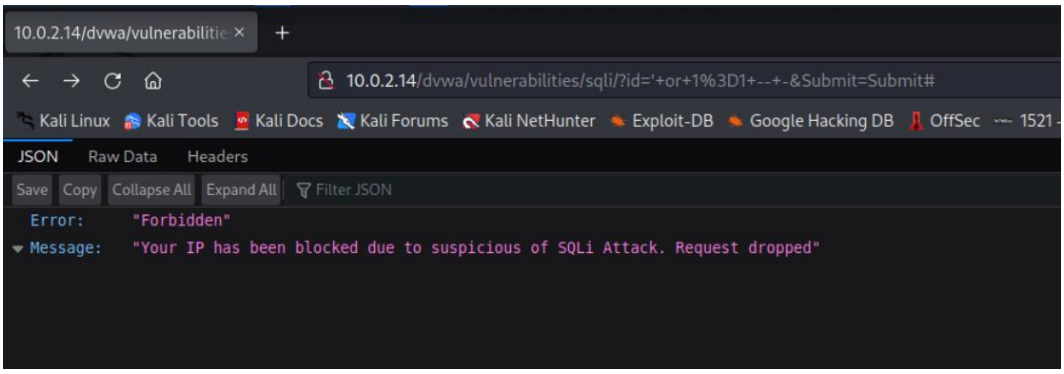


Figure 48 - SQLR34P3R Prevention through IP Blocking and Request Filtering

9.9 System Evaluation – Comparative Analysis

Framework	Detection Method	Data Source	Risk Analysis	Threat Modelling	Prevention

AMNESIA, 2005 [17]	Binary	HTTP Only	✗	✗	✓
Regular Expression, 2022 [18]	Binary	HTTP Only	✗	✗	✓
Aho-Corasick, 2022 [19]	Binary	HTTP Only	✗	✗	✓
SAFELI, 2008 [20]	Binary	HTTP Only	✗	✗	✗
SQLIFIX, 2021 [22]	Binary	HTTP Only	✗	✗	✗
Node centrality, 2016 [23]	Binary	HTTP Only	✗	✗	✗
Query Trees and Fisher Score, 2023 [24]	Binary	HTTP Only	✗	✗	✗
Multisource SQLi Detection, 2018 [25]	Binary	HTTP and Network Traffic	✗	✗	✗
Predictive machine learning 2020 [26]	Binary	HTTP Only	✗	✗	✓
SQLBlock 2020 [27]	Binary	HTTP Only	✗	✗	✓

CNN & MLP, 2021 [28]	Binary	HTTP Only	✗	✗	✓
CNN, 2019 [29]	Binary	Network Traffic	✗	✗	✗
LSTM, 2019 [30]	Binary	HTTP and Network Traffic	✗	✗	✗
LSTM/MLP, 2020 [31]	Binary	HTTP Only	✗	✗	✗
synBERT, 2023 [21]	Binary	HTTP Only	✗	✗	✗
PNN, 2023 [32]	Binary	HTTP Only	✗	✗	✗
SQLR34P3R	Multi	HTTP and Network Traffic	✓	✓	✓

Table 22 - State-of-the-Art Comparative Analysis