Detection of SQL injection using Reinforcement Learning

BACHELOR OF COMPUTER ENGINEERING

by

Tejas Sheth, 19102026

Nidhi Singh, 19102042

Janhavi Anap, 19102043

Het Patel, 19102005

Guide

Prof. Ramya R B

Department of Computer Engineering

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

2022-2023

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

CERTIFICATE

This is to certify that the project entitled "***Detection of SQL injection using Reinforcement Learning***" is a bonafide work of **"Tejas Sheth, (19102026), Nidhi Singh (19102042), Janhavi Anap (19102043), Het Patel(19102005)"** submitted to the University of Mumbai in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Engineering.**

Guide	Project Coordinator
Prof. Ramya R B	Prof. Rushikesh Nikam
Head of Department	Principal
Prof. Sachin Malave	Dr. Uttam Kolekar

Date:

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

Project Report Approval for B.E.

This project report for Sem-VII entitled ***Detection of SQL injection using Reinforcement Learning*** by **"Tejas Sheth, (19102026), Nidhi Singh (19102042), Janhavi Anap (19102043), Het Patel(19102005)"** is approved for the degree of ***Bachelor of Engineering*** in ***Computer Engineering***, ***2022-23***.

Examiner Name Signature

1. _____

2. _____

Date:

Place:

Declaration

We declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Tejas Sheth 19102026)

(Nidhi Singh 19102042)

(Janhavi Anap 19102043)

(Het Patel 19102005)

Date:

Abstract

The SQL Injection vulnerability can either be unintentional by software developers or an intentional ploy employed by a hacker with malicious intent to exploit sensitive data. With the recent surge in information, there is an innate quest to safeguard this information from falling into the wrong hand leading to data theft, leak of personal data or loss of property. With relational databases like MySQL being the most popular, it allows users to extract any available information without any significant knowledge of databases. With vast information stored in databases warrants attacker's attention, potentially risking critical confidential information. The premature detection of SQL Injection Attacks will be very helpful in preventing any malicious attempt by an attacker. In this research, we analyze the results of Reinforcement Learning algorithms like Q-Learning on a dataset consisting of potential SQL Injection queries. We intend to provide a Reinforcement Learning solution to minimise the potential threat posed by SQL Injection and give apriori to the model to learn to detect a SQL attack and prevent any unforeseen mishap more quickly and accurately.

Keywords*: ***Machine Learning, Information security, SQL injection, SQL detection, SQL injection Attacks, Reinforcement Learning, Q-Learning, Vulnerability detection, Apriori***

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abbreviation

SQLi	Structured query language injection
CIA	Confidentiality, Integrity, Availability
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network

CHAPTER 1

Introduction

SQL is a Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL is the standard language for Relational Database Systems. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language. SQL is widely popular because it offers the following advantages −

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create views, stored procedures, and functions in a database.
- Allows users to set permissions on tables, procedures and views.

Due to its wide popularity, it makes SQL more susceptible to attacks. SQL is designed to allow people access to information and is therefore inherently vulnerable. SQL is agnostic, meaning it works across database platforms. The upside to this is that it allows code to be database-server agnostic. But it is also the source of the problem. To prevent most vulnerabilities, developers should use parameterized SQL or stored procedures specific to the database server.

One of the big reasons why SQL injection maintains traction is due to improper development planning and the use of insecure development architecture. Making use of unsupported or legacy software or features introduces security holes that may not be patched or caught as quickly as they would with modern software. Running patched and modern versions of software are critical to avoiding security exploits, including SQL injection.

SQL injection is a web security vulnerability that allows an attacker to interfere with queries that an application makes to its database. This attack occurs at the application layer. Through successful SQL injection an attacker can view, modify, delete the data and, also get access to sensitive data. This leads to breach of the three security principles, CIA, i.e. confidentiality, integrity and authenticity of data. The main consequences are:

Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL Injection vulnerabilities.

Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.

Authorization: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL Injection vulnerability.

Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection attack.

This attack occurs when you ask for user input, like username, and the user incorrectly fills it with some SQL statement that gets unknowingly executed on one's Database. SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Because they are relatively easy to implement, and because the potential reward is great, SQL injection attacks are not uncommon.

Statistics vary, but it's estimated that SQL injection attacks comprise the majority of attacks on software applications. According to the OWASP (Open Web Application Security Project), injection attacks, which include SQL injections, were the third most serious web application security risk in 2021 [23].

1.1 SQL Injection

There has been an exponential rise in the number of dynamic websites. We have to handle large amounts of data stored for varied purposes, as well as their modification has to be fast to provide a

rich user experience. For this, we have to rely on relational databases like MySQL, MSSQL, etc. which are based on standard query language SQL.

SQL communication between the website and the SQL server consists of SQL queries. The most common and frequently used operation is data retrieval using the SELECT command with the WHERE clause, an advancement can be the use of multiple SQL queries concatenated with a UNION statement returning just one table.

SELECT column1, column2 FROM table1 WHERE column1=column2;

SELECT column1 FROM table1 WHERE column1=10 UNION SELECT column2 FROM table2 WHERE column2=column1;

A hacker manages SQL Injection whenever the server-side scripting has an inappropriate input validation, leading to the attacker gaining partial or full access to the query and thus the database.

SELECT column1 FROM table WHERE column2=user-input;

SELECT column1 FROM table WHERE column1=1 OR 1 = 1;

This was the most simple SQL Injection, more refined queries are those when the attacker adds a UNION statement and combines multiple queries from the original query. There are many types of conventional and modern SQL Injection attacks. They are primarily divided into

A. Classical SQLi (Basic SQLi) attacks are the simplest and most frequently used form of SQLi. These may occur when users are permitted to submit a SQL statement to a SQL database through user input. There are 4 different types: Piggy-Backed, Stored Procedures, Union Query, and Alternative Encoding according to Halfond, W. G. et. al. [18] and Wei, K. et. al. [19].

1) Piggy-Backed Queries: Instead of modifying the original query, the attacker intends to develop new queries that piggyback on it. As a result, the DBMS gets inundated with SQL queries. The first query is a standard query that is run normally, while the succeeding ones are run to complete the attack.

   e.g., normal SQL statement + ";" + INSERT (or UPDATE, DELETE, DROP) <rest of injected query>

2) Stored Procedure: When a conventional SQL query (such as SELECT) is generated as a stored procedure, an attacker can inject another stored procedure as a substitute resulting in a denial of service (DOS), or execute remote instructions.

   e.g., normal SQL statement + "; SHUTDOWN; " <rest of injected query>

3) Union Query: An attacker injects a UNION SELECT query to mislead the programme into

providing data from a table other than the one intended as stated by Hwang, D. (2022) [20].

e.g., normal SQL statement + "semicolon" + UNION SELECT <rest of injected query>

4) Alternative Encoding: The attacker modifies the SQLi pattern such that standard detection and prevention systems miss it. In this method, the attacker employs hexadecimal, Unicode, octal, and ASCII code encoding in the SQL Statement

B. Advanced SQLi

1) Blind SQLi: Attackers devised strategies to circumvent the lack of error notifications while still knowing whether the input is being treated as a SQL statement. This technique is often used in two variations: content-based blind SQLi and time-based blind SQLi.
2) Fast Flux SQLi : Fast Flux is a DNS method to conceal phishing and malware distribution sites behind a constantly changing network of compromised servers. The Asprox botnet was used to launch the large SQLi assault employing rapid flux. In Fast Flux mode, the DNS (Domain Name Server) hosts many malware-infected IPs at the same time, and the IPs rapidly change.
3) Compounded SQLi: A compound SQLi attack is a pair of two or more attacks that target the webpage and have far-reaching implications than the previous SQLi mentioned. The fast development of detection and mitigation measures for multiple SQLis has resulted in the emergence of compound SQLi. SQLi combined with DDoS attacks, DNS hijacking, XSS, and insufficient authentication are just a few examples.

CHAPTER 2

Literature Survey

SQL Injection (SQLi) is a type of attack in which an attacker inserts a malicious SQL query into the web application by appending it to the input parameters. Sadeghian, A. et. al. illustrate the classification of injection attacks like tautologies, illegal / logically incorrect queries, union queries, piggy-backed queries, stored procedures, inference, and alternate encodings [15]. Security researchers have categorized the solutions for SQLi into three main groups: Best code practices, SQLi detection and SQLi runtime prevention. The optimum solution would be writing secure code and among best code practices- parameterized querying is the most secure and efficient technique.

Rai, A. et. al. illustrate the classification and prevention of different SQLi attacks. SQLi is generally classified as In-band SQLi, Inferential SQLi and Out of Bound SQLi [16]. In-Bound SQL injection is further classified as Error-based and Union-based SQLi. Inferential SQLi can be broken down into Boolean-based Blind SQL and Time-based SQL. Defensive techniques that could be used to prevent an SQLi attack include Whitelisting/Blacklisting, prepared statement/ parameterized query, stored procedure, defensive coding practice, taint-based approach, proxy filters, instruction set randomization, low privileges and output Escaping. Different countermeasures work for different SQL Attacks.

Medhane and M. H. A. S. based their approach on SQLi grammar to identify the SQLi

vulnerabilities during software development and SQLi attack based on web-based applications [17]. The attacker's area unit used SQL queries for assaultive and hence these attacks reshare the SQL queries, thus neutering the behavior of the program.

John, A. proposed methods consisting of the best features of parse tree validation and code conversion techniques [4]. The algorithm parses the user input and checks whether it's vulnerable if any chance of vulnerability is found it applies code conversion over that input. Results show few drawbacks of code conversion as applying it to every user input is more time consuming and as well as the database also increases. The parse tree validation technique could raise a false alarm if a legitimate user is having blank space in his/her input. The proposed method proved to provide higher security levels than the individual techniques of code conversion and parse tree validation.

Hanmanthu, B. et. al. illustrates the use of the famous decision tree classification techniques to prevent SQLi attacks [5]. The proposed model works by sending different specially planned attack requests to the proposed SQLi decision tree model, and the final SQLi database is created for using classification data. It uses the satisfied analysis technique for finding the SQLi attack and uses the SQL decision tree. Software engineers usually rely on dynamic query building with string concatenation to construct SQL statements. The proposed method makes it possible to engineer different queries based on varying conditions set by users, without the need for manual interactions or error-prone code. The model showed consistency in attack detection and elimination at an average of 82% for all types of attacks. In order to perform a comparative evaluation of the proposed model, authors in [5] compared the proposed model to the other SQL scanning model which includes Acuneits, Netsparker, and Web cruiser and the results of the proposed model show good accuracy in comparison to other models.

Akinsola Jide et.al. gives us an idea about different types of SQLi attacks as already mentioned by Rai, A. et. al. [6][16]. The three main types are Classic In-band SQLI, Inferential Blind SQLI, and SQLi Based On Out-of-Band. They present the comparative analysis of different supervised ML algorithms to mitigate SQLi attacks. Besides precise accuracy and minimum errors, ML models also require putting several factors into consideration. The following metrics were taken into consideration to decide the effectiveness of the algorithm: Kappa Statistic, True Positive (TP) Rate, Accuracy, True Negative (TN) and (time to build the model (TTB)), for each of the machine learning algorithms.

Tang, P. et.al. only extracted and classified the URL features [7]. The factors like payload length, keywords and their weights are considered for feature extraction. The URL is classified as

malicious or non-malicious using ANN (Artificial Neural Network) models. The method and algorithm used here are multi-layer perceptron (MLP) and LSTM, both of which were implemented using Pytorch. The trained model is deployed in the ISP system so that abnormal behaviours can be found in the network in real-time. One of the drawbacks of using such an approach is that using the LSTM, model recognition is poor with high processing time & has lower accuracy.

Four machine learning models were considered in Kamtuo, K., & Soomlek, C. and they were compared, Support Vector Machine (SVM), Boosted Decision Tree, Artificial Neural Network, and Decision Tree [8]. They have proposed a framework using a compiler platform and ML to detect SQLi in queries which are illegal and logically incorrect on server-side scripting. The dataset consists of 1100 samples of vulnerable SQL commands. After training the model with the dataset it was evaluated in terms of probability of detection, probability of false alarm, precision, accuracy, and processing time. Decision Jungle was the best in performance showing results as the best machine learning model which related to the processing time of 2.4725 seconds and accuracy of 0.9968.

Ross, K. collected traffic from two points: a web application host and a Datiphy appliance node [9]. It is demonstrated that the accuracy obtained with correlated datasets using algorithms such as rule-based and decision-tree are nearly the same as those with a neural network algorithm, albeit with significantly improved performance.

Reinforcement Learning (RL) is known for obtaining knowledge by trial and error and continuously interacting with a dynamic environment. It is characterized by self-improving and online learning, making it one of the intelligent agents (IA) core technologies. The reinforcement signal provided by the environment in RL is to make a kind of appraisal of the action quality of the IA, but not tell the IA how to generate the correct action. The basic model of RL as stated in Qiang, W., & Zhongli, Z. includes a state, action and reward system [10]. Where the IA perceives the environment and chooses an action to obtain the biggest reward value by continuously interacting with the environment. The ultimate goal of RL is to learn an action strategy. The basic theory of reinforcement learning technology is: If a certain system's action causes a positive reward for the environment, the system generating this action lately will strengthen the trend, this is a positive feedback process; otherwise, the system generating this action will diminish this trend. Typical RL method based on the Markov decision-making process (MDP) model includes two kinds: Model-based methods such as the SARSA algorithm and Model-irrelevant methods,

such as the TD algorithm and the Q-learning algorithm.

Tian, W. et. al. illustrates methods to generate more effective penetration test case inputs to detect SQLi vulnerability [11]. The model-based penetration test method is found to generate test cases covering more types and patterns of SQLi attack input to thoroughly test the 'blacklist filter mechanism' of web applications. Here, the authors proposed two-step penetration test case generation, building and instantiating, where step 1 reveals what test case should be used while step 2 expounds on how many test cases should be used. This study focuses on the adequacy of penetration test case inputs for the SQL injection vulnerability. It builds an experimental platform to verify the proposed test case generation methods.

Ghanem M. C., & Chen T. M. proposes and evaluates an AI-based pentesting system which makes use of RL to learn and reproduce average and complex pentesting activities [12]. The scope is limited to network infrastructures PT planning and not the entire practice. Moreover, the authors tackle the complex problem of expertise capturing by allowing the learning module to store and reuse PT policies in a more efficient way.

Niculae, S. et al. measured the performance of multiple fixed-strategy and learning-based agents [13]. They concluded that Q-learning, with some extra techniques applied and greedy agent initialisation, performed best, surpassing human performance in the given environment.

Hu, Z., Beuran, R., & Tan, Y. suggests an automated penetration testing framework, based on deep learning techniques, particularly deep Q-learning networks (DQN) [14]. The authors conducted an experiment in which a given network host was populated with real host and vulnerable data, to determine the optimal attack path, and to provide viable solutions.

Erdődi, L. et. al. simplified the dynamics of SQLi vulnerabilities by casting the problem as a security capture-the-flag and implementing it as an RL problem [1]. Assuming that the vulnerability has been identified, they rely on RL algorithms to automate the process of exploiting SQLi. They implemented the model using two simulations. The first simulation showed that a simple RL agent based on a Q-learning algorithm can successfully develop an effective strategy to solve the SQLi problem. A tabular Q-learning algorithm can discover a meaningful strategy by pure trial and error and can reach a performance close to the theoretical optimum. Using a table to store the Q-value function allowed them to carry out a close analysis of the learning dynamics of the agent, but this approach had poor scalability. Thus, in the second simulation, they sacrificed interpretability in order to work around the issue of scalability. They deployed a deep Q-learning agent to tackle the same problem as in the first simulation. The deep Q-learning agents were able

to learn a good strategy for the SQLi problem as well as provide a solution to the space constraints imposed by the instantiation of an explicit Q-table.

Given the success of RL in tackling and solving games, Penetration Testing, when distilled as a capture-the-flag (CTF), can be expressed as a game. However, in the case of penetration testing, an artificial agent may learn only by trial and error while a human hacker may rely on alternative sources of knowledge, deductions, hypothesis testing, and social engineering. Although an RL agent may in principle learn the structure from scratch in a pure model-free way, this may turn out to be a computationally hard challenge. Thus according to Zennaro, F. M., & Erdodi, L.injecting some form of elementary a priori knowledge about the structure of the problem may simplify the learning problem [2]. Some basic forms of apriori knowledge which make the RL agent more efficient are lazy loading, state aggregation and imitation learning. The authors categorized CTFs in groups according to the type of vulnerability they instantiate and the type of exploitation that a player is expected to perform. The prototypical classes of CTF problems considered were port scanning and intrusion, server hacking and website hacking. All the simulations were implemented using the standard RL interface defined in the OpenAI gym library. Simulation 1 solved the Port Scanning CTF problem using the basic tabular Q-learning algorithm. Solving this challenge required learning the problem meaning that the RL agent has to rely strongly on exploration. Simulation 2 solved the Non-stationary Port Scanning CTF problem by extending the previous problem by considering a more challenging scenario in which the target system is not stationary, but it may randomly change in response to the actions of the agent. Introducing non-stationary dynamics made the problem more challenging by preventing the agent from learning the exact structure of the problem with certainty. Despite this, the Q-learning agent was still able to solve the CTF problem in a reasonable yet sub-optimal way. Simulation 3 solved the Server Hacking CTF problem with Lazy Loading which considers a more realistic scenario. The problem presented a serious challenge to the tabular Q-learning agent because of the size of its Q-table. Relying on a priori knowledge in the form of lazy loading controlled the dimensionality of the state and action state pruning the non-relevant states. This method allowed the agent to discriminate between relevant and non-relevant states based on its experience. Simulation 4 solved the Website Hacking CTF problem with State Aggregation preserving most of the complexity of Simulation 3. State aggregation allowed them to inject useful prior information about the structure of the problem, thus simplifying exploration and reducing the number of (state, action) pairs. Simulation 5 solved the Web Hacking CTF problem with Imitation Learning which

emulates learning in a teacher-and-student setting, where expert paradigmatic behaviors are offered to a student to speed up its learning. Imitation learning proved to be an effective technique to enable faster learning for the RL agent. The improvement was due to the possibility of introducing the agent's knowledge of the structure of the problem. Instead of encoding knowledge of the structure of the problem in a formal mathematical way, they provided the RL agent with concrete observations about the structure of the problem. The agent could successfully exploit this information in order to learn an optimal policy.

Verme, M. D. et. al. considered the problem of exploiting SQLi vulnerabilities, representing it as a capture-the-flag scenario in which an attacker can submit strings to an input form with the aim of obtaining a flag token representing private information [3]. The attacker was modeled as an RL agent that interacts with the server to learn an optimal policy leading to an exploit. The authors did a comparison between two types of agents, one was a simple structured agent that relied on significant a priori knowledge and used high-level actions and the other was a structureless agent that had limited a priori knowledge and generated SQL statements. The comparison showcased the feasibility of developing agents that relied on less ad-hoc modeling.

Author	Title	Source	Findings
Erdődi, L., Sommervoll, Å. Å., & Zennaro, F. M. (2021)	Simulating SQL Injection Vulnerability Exploitation Using Q-Learning Reinforcement Learning Agents[1]	https://arxiv.org/pdf/2101.03118.pdf	Assumed that the injection was detected and automate the process of exploiting SQLi using tabular Q-learning and deep Q-learning of the Reinforcement Learning
Zennaro, F. M., & Erdodi, L. (2020)	Modelling Penetration Testing with Reinforcement Learning Using Capture-the-Flag Challenges[2]	https://arxiv.org/pdf/2005.12632.pdf	It uses similar method as the above paper but also provides some a priori knowledge to the model to improve results. It also explored situations like Port scanning, Server Hacking, and Web Hacking
Verme, M. D., Sommervoll, Å. Å., Erdődi, L., Totaro, S., & Zennaro, F. M. (2021, November)	SQL Injections and Reinforcement Learning: An Empirical Evaluation of the Role of Action Structure[3]	https://link.springer.com/chapter/10.1007/978-3-030-91625-1_6	This paper did comparison between the performance of simple structured agents and agents which were provided with a priori knowledge.

Table 2.1: Survey of Literature

CHAPTER 3

Limitation of Existing system

Supervised learning algorithms have been widely used in detecting SQL injection attacks. However, there are some limitations of existing systems that use supervised learning for detecting SQL injection attacks. Some of these limitations include:

1) Limited Training Data: The accuracy of a supervised learning system is highly dependent on the quality and quantity of the training data. The limited availability of high-quality training data can lead to a high rate of false positives and false negatives.
2) Evolving Attack Patterns: Attackers are constantly evolving their techniques to evade detection by existing systems. Supervised learning systems can become less effective over time as attackers develop new attack patterns that are not captured in the training data.
3) Overfitting: Overfitting occurs when a machine learning model is trained on a limited dataset and becomes too specialized to that dataset. This can lead to poor performance when the model is applied to new data that it has not been trained on.
4) Lack of Contextual Information: Supervised learning algorithms may not take into account contextual information, such as the user's behavior, which can lead to false positives.
5) Limited Scalability: Supervised learning systems can be limited in their ability to scale to large datasets or high-volume traffic.

CHAPTER 4

Problem Statement, Objectives and Scope

4.1 Problem Statement

Given a web application, the goal is to detect whether a given input has an SQL injection attack appended to it or not. The RL agent should learn to take actions based on its observations of the web application's behavior and the results of its previous actions to minimize the number of false positives and false negatives. The agent's actions can include modifying the SQL queries sent to the database or blocking suspicious requests. The RL agent's performance will be evaluated based on metrics such as precision, recall, and F1-score. The system should be able to adapt to new attack patterns and learn from its mistakes to continuously improve its performance.

4.2 Objectives

- Create a basic Web Application to perform SQLi attacks.
- Detection and Prevention of SQLi Attacks using Reinforcement Machine Learning Agents.

4.3 Scope

So far existing methods have considered dealing with SQLi by identifying and classifying the vulnerabilities and relied on supervised or unsupervised machine learning algorithms. The currently available datasets work effectively on tautology queries, but lack the means to detect the compounded queries. The problem of performing SQL injection can be considered a challenge that can be tackled with reinforcement learning methods. The reinforcement learning agents can help to identify the SQLi attack and prevent it from being executed, thus improving the security of the website by detecting the injection before executing the query in the backend.

CHAPTER 5

Proposed System

5.1 Proposed System Overview

SQL Injection continues to be one of the most noxious security exploits widely used by hackers all over the world. The vulnerability can be unintentional by software developers during the development phase, an intentional ploy employed by a hacker to exploit or corrupt personal sensitive data or for unknown reasons. The exploitation leads to the breach of CIA principle i.e. Confidentiality, Integrity and Availability.

In this project, we aim to use the Reinforcement Learning method towards SQLiA detection. Here, in the training phase, we would provide apriori to the Q-Learning model. The training dataset would contain both malicious and non-malicious queries. Providing efficient apriori should be the most crucial factor in order to obtain higher efficiency.

5.2 Design Details

Architecture Diagram

An architectural diagram is a visual representation that maps out the physical implementation for

components of a software system. It shows the general structure of the software system and the associations, limitations, and boundaries between each element.

The architecture of the project includes two databases, a detection model and a user interface. There are two databases, one for storing user details and one for storing the training SQL injection training data. The SQL injection detection model refers to the database having SQL injection to train itself and predict whether the input is an injection or not.

The User interface comprises the signup and login section. The signup section would register the user whereas the login section would be the interface containing the input field through which actual login and injections can be passed for SQL injection detection.

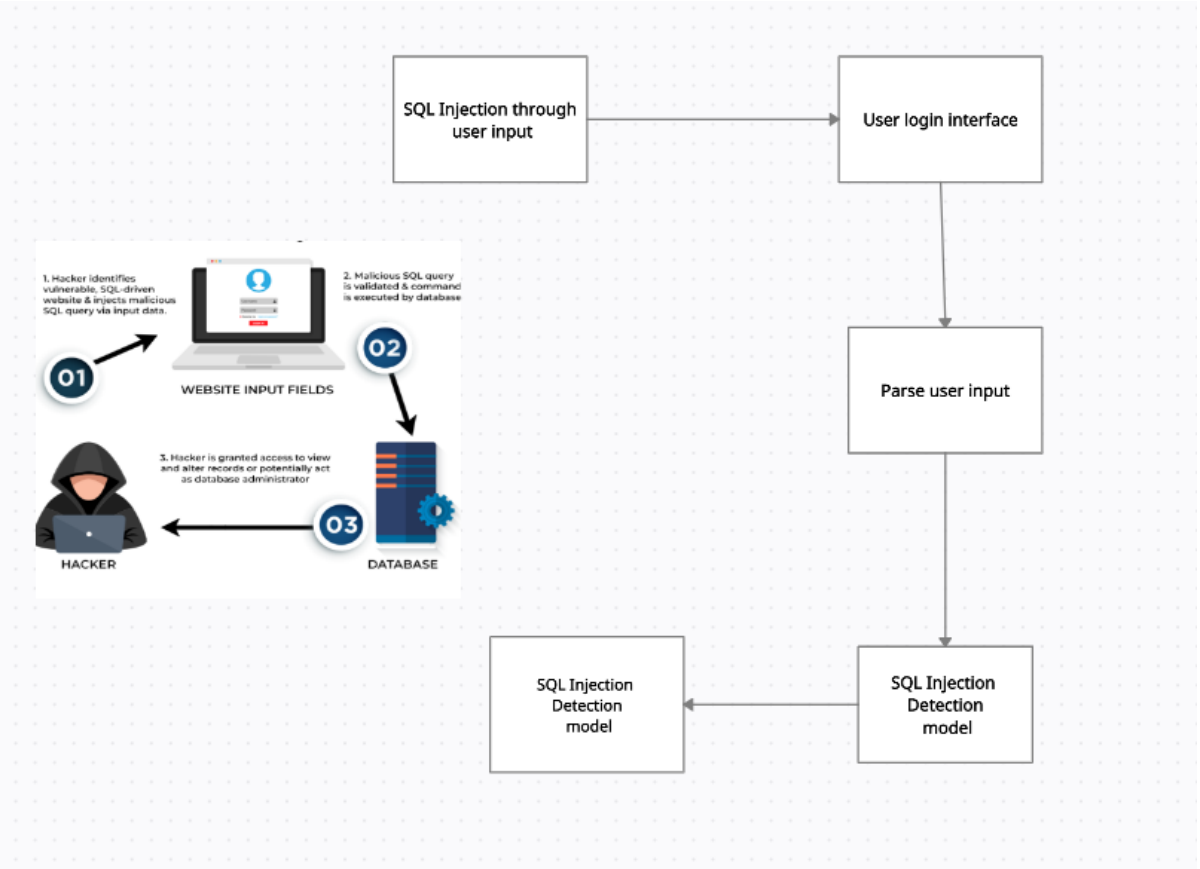


Figure 5.1.1: Architecture Diagram

Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an

information system , modeling its process aspects. Often it is a preliminary step used to create an overview of the system that can later be elaborated.

Level-0 Data Flow Diagram

The general flow of the program is that the user interacts with our web page The user is asked to login if it is a valid user then they can enter their credentials and log in to the system. If the user is with malicious intent or attacker then they can perform SQL injection

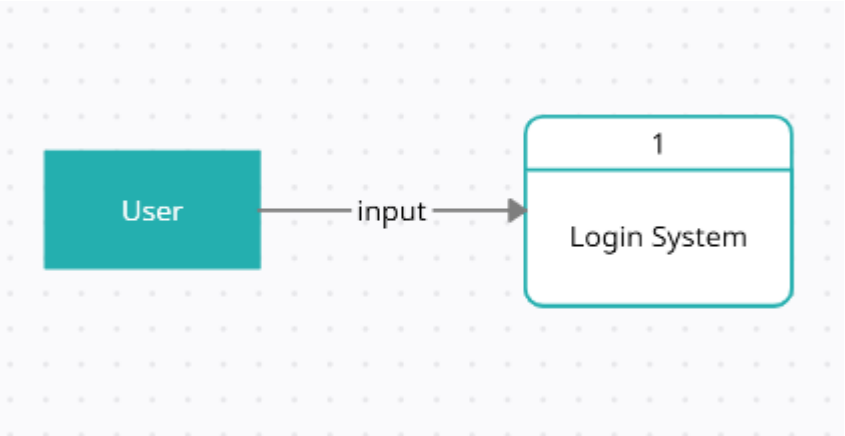


Figure 5.1.2: Data Flow Diagram (level 0)

Level-1 Data Flow Diagram

If a new user interacts with the Login System they can create a new account by providing details and signing up. By signing up, their details will be stored in the user database. If an existing user wants to log in then they can login by providing input through the login interface. The user has to provide username and password as input data which is checked for any SQL injection. If the password by the user matches that in the user database then it is a successful login else it is unsuccessful. If a user inputs an injection then it will be detected by the SQL injection model which is trained over dataset containing SQL injection and display an “attack detected” message

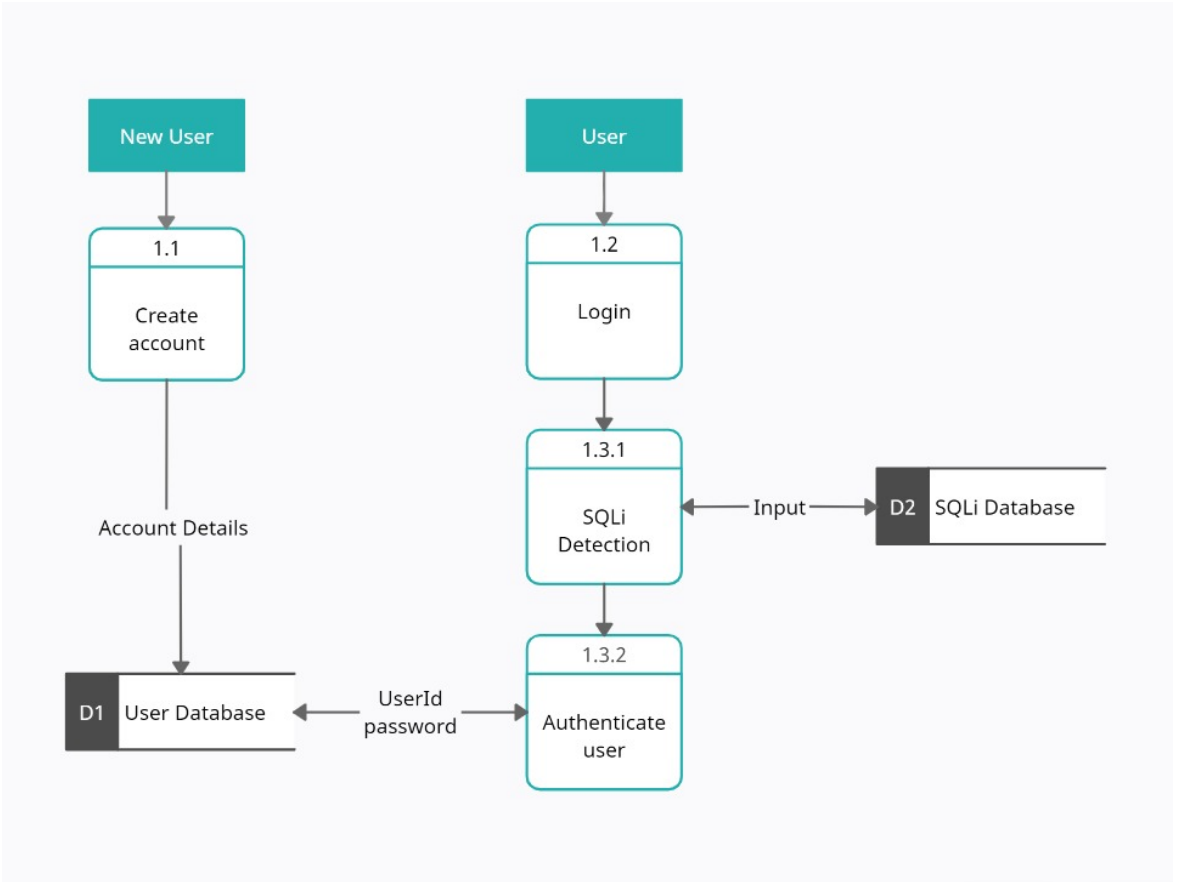


Figure 5.1.3: Data Flow Diagram (level 1)

Sequence Diagram

A sequence diagram is a Unified Modeling Language (UML) diagram that illustrates the sequence of messages between objects in an interaction. A sequence diagram consists of a group of objects that are represented by lifelines, and the messages that they exchange over time during the interaction.

The sequence diagram with respect to the user can be explained as follows. First, the user provides an input through the user interface implemented on the web browser. In the backend, input is sent to the SQL injection detection model. If the model classifies the input as injection then an SQL injection attack detected message will be shown to the user. If the injection is not classified as SQL injection then it consults the user database and matches the password for the input user. If the password is matched then successful login message is displayed else unsuccessful login message is displayed.

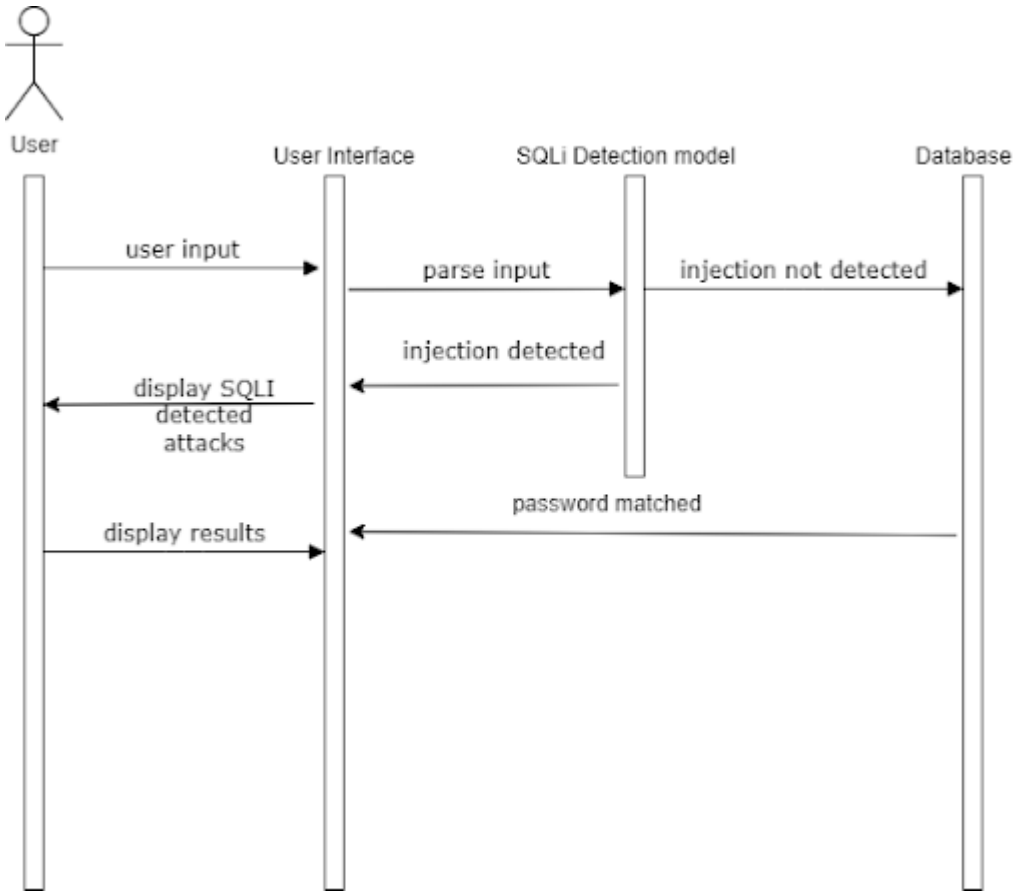


Figure 5.1.4: Sequence Diagram

Activity Diagram

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.

The user will input the data through the login interface and the detection model will check if the input is an injection or not. If the input is classified as an injection then a message displaying that an injection was detected will be shown. If the input is not classified as an injection then the login credential will be checked from the user database and will show if the login was successful or not.

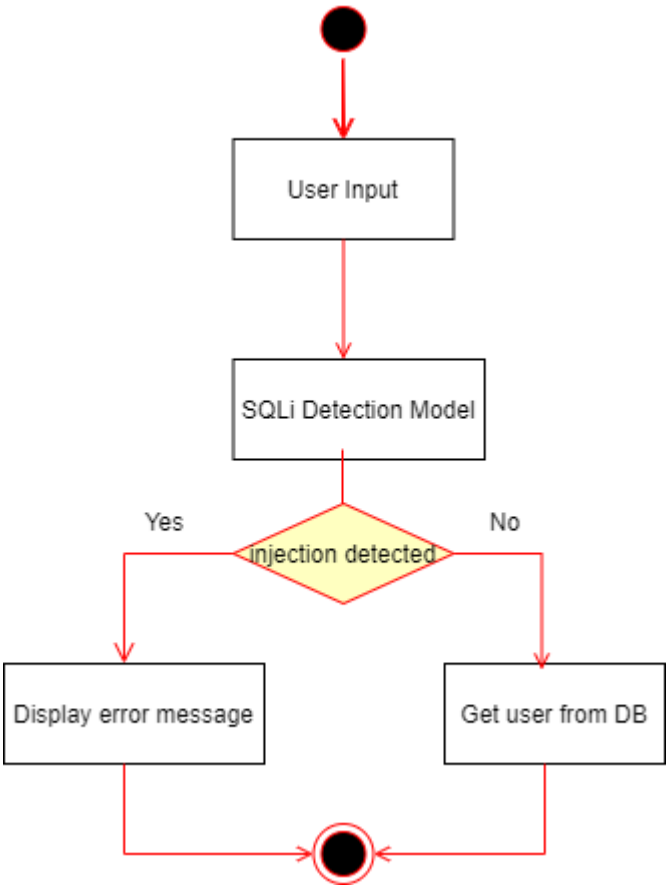


Figure 5.1.5: Activity Diagram

5.3 Methodology

The user provides an input through the user interface implemented on the web browser. In the backend, input is sent to the SQL injection detection model. If the model classifies the input as injection then an SQL injection attack detected message will be shown to the user. If the injection is not classified as SQL injection then it consults the user database and matches the password for the input user. If the password is matched then successful login message is displayed else unsuccessful login message is displayed.

Q-Learning is a model-free reinforcement learning algorithm.The Q-learning algorithm continues to iterate until the Q-values converge to the optimal values. The optimal policy can be obtained by selecting the action with the highest Q-value for each state

We have studied the various supervised and unsupervised algorithms used for the detection and prevention of SQL injection attacks. A comprehensive dataset was created considering all the data types of SQLi attack queries. We have compared multiple models used for detection like Naive Bayes, Logistic Regression, Random Forest, SVM, Decision Tree, and CNN. CNN and Naive Bayes models are found to show the most consistent accuracy and F1 scores. CNN model has a precision and F1 score of 97.26% and 94.85% respectively. The Naive Bayes model has a precision and F1 score of 99% and 94.79% respectively.

Since these results are obtained after training the model on an exhaustive dataset, we can conclude that CNN and Naive Bayes can prove to be a good a priori for the Reinforcement Learning model which could improve the performance measures.

5.3.1 Working of Models

1)Random Forest:

Random forests is an ensemble learning method for classification and regression tasks that constructs multiple decision trees at training time, corrects for decision trees' overfitting, and outputs the class selected by most trees for classification tasks and the mean prediction of individual trees for regression tasks.

2)Support Vector Machine:

Support Vector Machine (SVM) is a popular supervised learning algorithm used for classification and regression problems, that aims to create the best decision boundary or hyperplane to segregate n-dimensional space into classes using support vectors.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as a Support Vector Machine.

3)Decision Tree Classifier:

Decision tree is a supervised learning technique used for classification problems that follows a tree-structured classifier where internal nodes represent the features, branches represent decision rules and each leaf node represents the outcome, and predicts the class of the given dataset by comparing the values of the root attribute with the record attribute and following the branch to the next node.

This algorithm compares the values of the root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node. For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf node of the tree.

4)KNN

KNN is a algorithm which requires no parameters and stores all the available data and classifies a new data point based on similarity This method is more effective when the training data is large.

5)Convolutional Neural Network:

Convolutional Neural Network is a deep learning technique that uses weights and biases to distinguish distinct aspects/objects from one another. A CNN requires the least amount of pre-processing when compared to the other models in this research. A CNN combined with an Intrusion Detection System (IDS) allows us to analyze traffic and make educated judgments. This enhances accuracy and outcomes, and the frequency of false warnings may be greatly decreased.

The accuracy of CNN on the test set was 0.9726 and the F1 Score of CNN on the test set was 0.9485.

6) Reinforcement Learning:

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents

should take an action in an environment to maximise the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning. Reinforcement learning differs from supervised learning in not needing labeled input/output pairs to be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead, the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge)

Following are the components of Reinforcement Learning:

- Agent- entity that interacts with the environment and makes decisions based on the feedback it receives.
- Environment- a world from which the agent operates and receives feedback.
- State- represents the environment at a given time, which the agent uses to make decisions. State Space- set of all possible states the agent could take to achieve the required goal.
- Action- decision that the agent makes based on the current state of the environment.
- Action Space- a Finite set of possible actions that the agent can take.
- Reward- is an evaluation parameter returned from the environment to the agent.
- Policy- strategy applied by an agent for the next action based on the current state.
- Value- compared to the short-term reward, is the expected long-term return with a discount.
- Q-Value- similar to the value, but it takes one additional parameter known as current action (a). It is the expected cumulative reward of taking an action in a particular state and following a particular policy

Popular Reinforcement Learning algorithms:

1. State Action Reward State action (SARSA)

The one major variance between Q-Learning and SARSA algorithms is that, unlike Q-Learning, the maximum reward for the next state is not required for updating the Q-value in the table. The action that the agent takes would be to either label the query as legitimate or malicious. The reward signal would be based on the accuracy of the agent's labelling. During training, the agent uses SARSA to learn a policy that maximizes the expected future reward. The agent starts in an initial state and selects an action based on its policy. It then observes the reward and the new state resulting from the action and updates its Q-value based on the SARSA update rule. Once the agent has learned a policy through training, it can be used to detect SQL injection attacks in real

time. The agent would observe a new SQL query and use its policy to label the query as legitimate or malicious.

Compared to Q-learning, SARSA is an on-policy algorithm, which means it learns a policy while taking into account the current policy being used. This can be useful in situations where the agent needs to balance exploration and exploitation in a more controlled manner, and where the reward signal may be noisy or delayed.

Unfortunately, SARSA is known to converge slower than the Q-learning algorithm which can be an issue when training the algorithm on large datasets of SQL queries. SARSA has several hyperparameters that need to be carefully tuned to achieve optimal performance. This can be a time-consuming and challenging task, especially when dealing with large datasets.

2. Deep Q-Learning

It is a combination of reinforcement learning with deep neural networks, which allows for more complex representations of the state and action spaces. Deep Q-Learning (DQL) is a variant of Q-Learning that uses a neural network to approximate the Q-function, rather than a lookup table. During training, the agent uses DQL to learn a policy that maximizes the expected future reward. The agent observes a state i.e., an SQL query, and passes it through a neural network to estimate the Q-values for each action i.e., legitimate or malicious. The agent selects the action with the highest Q-value based on an epsilon-greedy policy. The agent then observes the reward and the new state resulting from the action and updates the neural network using the back-propagation algorithm. Once the agent has learned a policy through training, it can be used to detect SQL injection attacks in real time. The agent observes a new SQL query, passes it through the neural network to estimate the Q-values for each action, and selects the action with the highest Q-value. The agent then labels the query as legitimate or malicious based on the selected action.

Compared to traditional Q-Learning, DQL can handle high-dimensional state spaces more efficiently by using a neural network to approximate the Q-function. However, DQL requires a large amount of computation, particularly when using large neural networks. This can be a significant disadvantage when training the algorithm on large datasets of SQL queries. DQL can also be susceptible to overfitting, particularly when the dataset is small or noisy. Overfitting can lead to poor generalization to new SQL queries, reducing the algorithm's ability to detect SQL injection attacks.

3. Q-Learning:

Q-Learning is a model-free reinforcement learning algorithm. It is also known for its ability to capture complex relationships within input features and to handle non-stationary environments where the underlying distribution of the data may change over time. In this research, temporal difference learning has been used.

It is an algorithm that chooses the best action that should be taken at any possible state by estimating the value of each action known as a value-based algorithm. In Q-Learning, the agent maintains a Q-table which contains the estimated Q-values for each state-action pair. On each step, the agent selects an action to be performed based on its current state and updates the Q-table based on the observed reward and the estimated Q-values of the next state. Over time, as the agent experiences more states and updates its Q-table, the estimated Q-values converge to the true optimal Q-values, permitting the agent to learn the optimal policy. Thus, the algorithm guarantees to obtain optimal solutions.

Q-Learning guarantees global convergence to an optimal policy under certain conditions, which can provide more confidence in the learned policy than SARSA or DQL. This makes Q-Learning a more reliable option for SQL injection detection, where the consequences of false positives and false negatives can be severe.

5.3.1.1 Working of Q-Learning

Implementation of Q-Learning

Algorithm:

1. Initialize the Q-table with arbitrary values.
2. Observe the current state, s, of the system
3. Select an action, a, based on the current state and the Q-table, using an exploration strategy.
4. Take the action, a, and observe the next state, s', and the reward, r.
5. Update the Q-value for the current state-action pair using the observed reward and the maximum predicted Q-value for the next state
6. Set the current state, s, to the next state, s', and repeat steps 3-5.
7. Repeat the process until a stopping condition is met
8. Use the final Q-table to make predictions by selecting the action with the highest Q-value for each state.

In Reinforcement Learning, the Apriori algorithm can be used to identify patterns in the interactions between an agent and its environment. They help the agent optimize its decision-making and improve its overall performance. If the agent frequently takes a certain action when it is in a specific state, the Apriori algorithm identifies this pattern and suggests that the agent take this action in the future whenever it is in a similar state. This helps the agent avoid

sub-optimal decisions and improve its overall reward.

In our model, we provided CNN model as an apriori to the Q-learning algorithm. The CNN model was chosen over any other model like Naive Bayes because of its better ability to recognize a pattern in the input data that indicates the presence of an SQL injection attack. The variation in SQL injections is subtle, such as changes in the order of parameters or the use of different operators. CNN models are robust to these variations, as they can learn to recognize patterns regardless of their location in the input data. CNN models also have the ability to learn complex patterns that are composed of simpler patterns. And most importantly CNN models are computationally efficient and can be trained on large datasets. This is important for SQL injection detection, as the model needs to be trained on a large and diverse set of input data in order to generalize well to new and unseen attacks.

Apriori can be integrated into a Q-Learning model at various stages of the learning process depending on the requirements. Apriori can be integrated at the pre-processing stage when identifying frequent patterns or correlations in the data and transforming the data into a more suitable format for the Q-Learning algorithm is crucial. Apriori can be used in the exploration/exploitation trade-off stage to balance the need for exploration with the need for exploitation and ensure that the Q-Learning algorithm is able to learn effectively from the available data. Apriori can also be integrated into the Q-Value update stage where certain state-action pairs are prioritized over others to improve the accuracy and efficiency of the Q-learning algorithm.

Thus, we will be integrating the Apriori of the CNN model in the Q-Value update stage. The CNN model will provide robustness while providing apriori at the Q-value update stage, which will improve the accuracy. In a Q-Learning environment model, a step function is a function that defines the behaviour of an agent as it interacts with its environment. The step function is used to update the Q-Value estimates for each state-action pair based on the observed reward and the predicted reward for the next state. Integrating the Apriori algorithm into the step function of a Q-Learning model involves using the interpretations of the apriori (CNN) model to modify the Q-Value estimates and the algorithm of the CNN model added as apriori to the step function of a Q-Learning environment model remains similar to the standard algorithm of Q-Learning model. The only difference is in the definition of the step function.

The Q-value for a state-action pair (s, a) is updated using the observed reward after taking action a in state s and the predicted future rewards. The update is done using the following equation:

$$Q(s, a) = Q(s, a) + \alpha * (r + \gamma * \max(Q(s', a')) - Q(s, a))$$

where, Q(s, a) is the current estimate of the Q-value for state-action pair (s, a).

α, is the learning rate, which determines the extent to which new information overrides the old estimate of the Q-value.

r is the reward observed after taking action a in state s.

γ, is the discount factor, which determines the importance of future rewards relative to the present reward.s

Q(s', a') is the estimated Q-value for the next state s' and the best action a' in that state.

5.3.2 Implementation

This is the user interface on visiting the website

Fig 5.3.2.1: User interface

Once the user enters the credentials, the input is sent to the injection dataset to check if the entered string belongs to any class of injection. If it is not an injection, then the data is sent to our user database to check whether the user exists in our system.

Accordingly, the message is displayed.

Fig 5.3.2.2: Successful login

If the entered data is an injection , then following message is displayed

Fig 5.3.2.4: Injection detected

5.3.3 Results and discussion

We have studied the various types of algorithms used for the detection of SQL injection attacks. A comprehensive dataset was used considering different types of SQL injection queries. We have compared multiple Machine Learning models like SVM, Decision Tree, Random Fores, CNN and KNN as shown in Fig. 5.3.3.1. Random Forest and CNN model have been generating the most consistent accuracy and F1 scores.

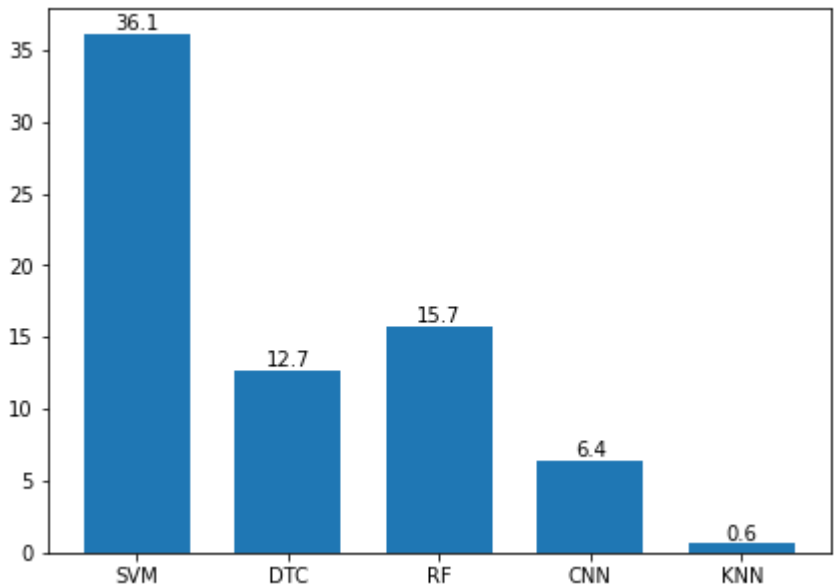


Fig 5.3.3.1: Comparison of the models

Fig. 5.3.3.2 shows the time required to compile the models. CNN and KNN took the least time to compile. The comparative results of KNN were quite satisfactory. Although KNN compiled the quickest, it detected more False Positives than the potential SQL injection attack queries which is an undesirable outcome as it can be seen in Fig.5.3.3.3.

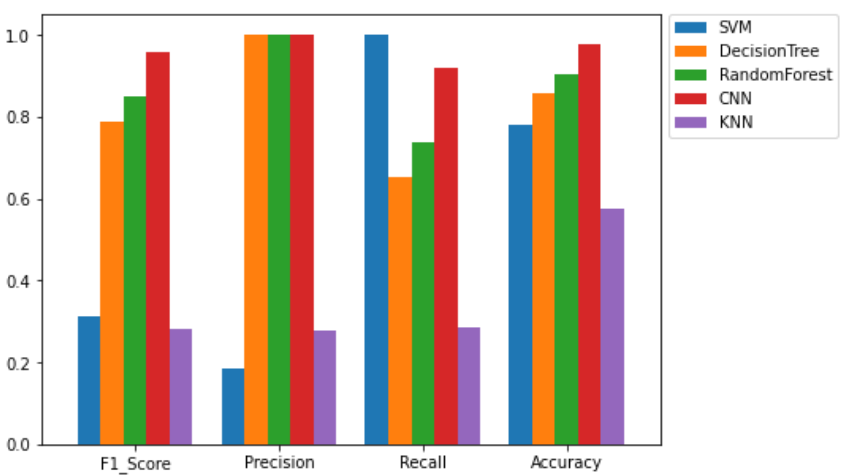


Fig 5.3.3.2: Compile time for the models

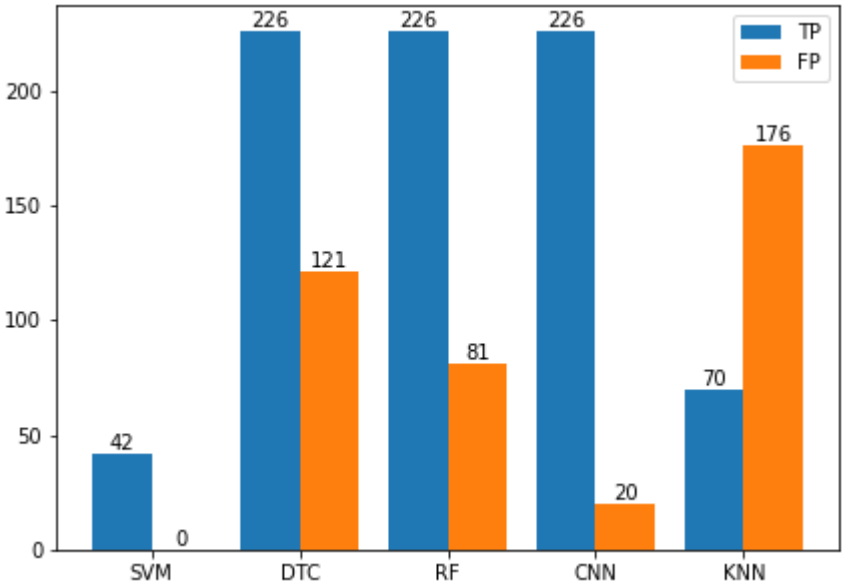


Fig 5.3.3.3: True Positive v/s False Positive

The performance measures of SVM, Decision Tree Classifier, Random Forest, CNN and KNN were compared. While CNN model shows consistent performance measures, the time required to build the CNN model is much more compared to that of KNN model method. Since these results are obtained after training the model on an exhaustive dataset, we can conclude that CNN can prove to be a good model comparatively that can detect SQL injection attack.

In THIS SEMESTER:

The learning rate is a trade-off between the algorithm's stability and its ability to respond to changes in the environment. It is a scalar value that ranges between 0 and 1. If the learning rate is high, the agent quickly assimilates new information and responds rapidly to the changes in the environment. However, a high learning rate makes the algorithm more unstable, since the agent overreacts to noisy information as seen in Fig.5.3.3.4. If the learning rate is low, the agent updates its estimates more slowly and is less sensitive to changes in the environment. However, a low learning rate can also make the algorithm slow to meet to the optimal policy, since it takes longer for the agent to learn from its experiences.

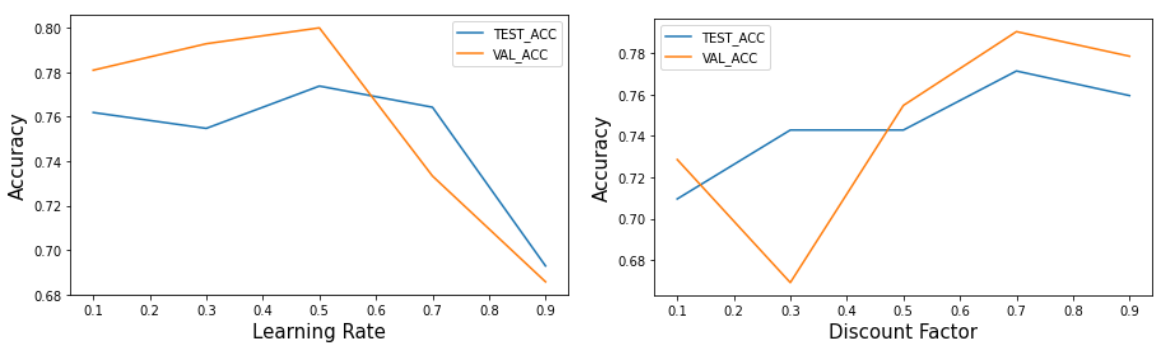


Fig 5.3.3.4

The discount factor influences the Q-Learning algorithm's balance between short-term and long-term rewards. It is a scalar value that ranges between 0 and 1. A high discount factor (close to 1) means that future rewards are considered

to be very important, and the agent will prioritize them over immediate rewards. A low discount factor (close to 0) means that the agent only cares about immediate rewards and does not consider future rewards as seen in Fig.5.3.3.5.

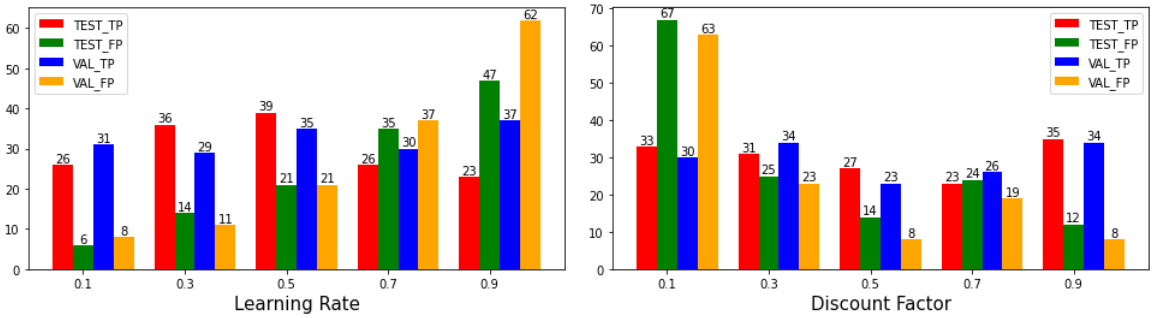


Fig 5.3.3.5

CHAPTER 6
Conclusion

Using a Q-Learning algorithm for detecting SQL injection attacks in Web Applications holds promise in achieving accurate and real-time detection while minimizing false positives and negatives. The algorithm can be trained on a dataset of SQL queries and their labels to learn a policy for detecting malicious queries. Further research in this area can lead to the development of more sophisticated and robust algorithms for securing web applications against SQL injection attacks. To increase the accuracy of the Q-Learning model, it is suggested to use a learning rate closer to 0 and a discount factor closer to 1 as seen from Fig.\ref{accLR} and Fig.\ref{accDF} where the accuracy for validation data is greater than the accuracy for testing data in an ideal condition. This can increase the accuracy from 66\% to 76\%. As the model learns from the training data, further learning with testing and validation data can lead to an increased accuracy from 70\% to 76\%. With a learning rate of 0.1 and discount factor of 0.9, not only can better accuracy values be obtained, but the difference between the number of actually detected injections and false alarms can also be maximized as seen from Fig.\ref{LRTPFP} and Fig.\ref{DFTPFP}.

CHAPTER 6

Experimental Setup

6.1 Software Requirements:

1) **Python:** General Programming Language used to code the model, includes several libraries.
2) **Python libraries:** Numpy, Matplotlib, PIL (Python Imaging Library), pandas, Convolutional CNN.
3) **VS Code:** IDE to run, train and test the ML model. Plenty of extensions, open-source, cross-platform support,
4) **Git & GitHub:** Version Control System used for collaboration.
5) **Flask:** Flask is used for developing web applications using python, implemented on Werkzeug and Jinja2. The advantage of using the Flask framework is that here is a built-in development server and a fast debugger is provided.
6) **Google Workspace:** Project Management Tool to improve team coordination and file sharing.
7) **Teamgantt:** Project Management Tool to improve team coordination and file sharing using the Gantt chart.

6.2 Hardware Requirements:

1) **CPU:** 1.8 GHz or faster 64-bit processor; Quad-core or better recommended.
2) **Processor:** Intel i5 or greater.
3) **RAM:** Minimum of 4GB of ram.
4) **Storage:** 4GB of free hard disk space.

CHAPTER 7

Project Plan

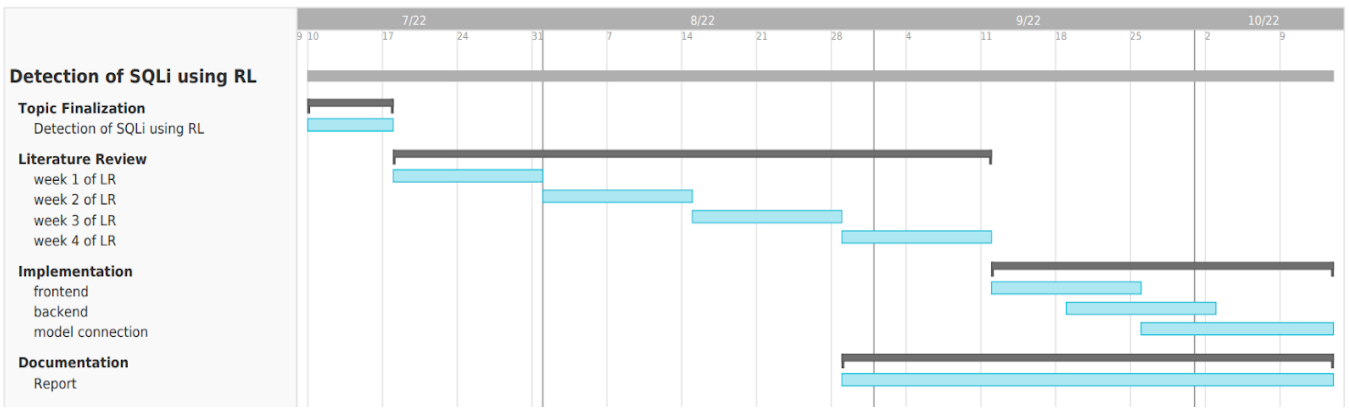


Fig 7: Gantt Chart

CHAPTER 8
Future Scope

The project is able to detect majorly the tautology queries and some complex queries.The injection on variety of databases cannot be predicted as for now due to limitation of the training dataset.As the dataset keeps on increasing,simultaneously the accuracy can also be increased.

An SQL detection machine learning system can be used to automate the testing of database queries and ensure that they are functioning correctly.

It can be further implemented to create an external plug-in system to act as an added security layer on an SQL database.

It can be deployed to monitor queries to the SQL database in real-time.

SQL injection attacks are constantly evolving, so it's important to regularly update the SQL detection system to recognize new attack patterns and adapt to changing threats.

References

[1] Erdődi, L., Sommervoll, Å. Å., & Zennaro, F. M. (2021). Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents. Journal of Information Security and Applications, 61, 102903.

[2] Zennaro, F. M., & Erdodi, L. (2020). Modeling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge. arXiv preprint arXiv:2005.12632.

[3] Verme, M. D., Sommervoll, Å. Å., Erdődi, L., Totaro, S., & Zennaro, F. M. (2021, November). SQL Injections and Reinforcement Learning: An Empirical Evaluation of the Role of Action Structure. In Nordic Conference on Secure IT Systems (pp. 95-113). Springer, Cham.

[4] John, A. (2015). SQL Injection Prevention by adaptive algorithm. IOSR journal of computer engineering, 17, 19-24.

[5] Hanmanthu, B., Ram, B. R., & Niranjan, P. (2015). SQL Injection Attack prevention based on decision tree classification. 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO).

[6] Akinsola Jide, E. T., Oludele, A., & Idowu Sunday, A. (2020). SQL injection attacks predictive analytics using supervised machine learning techniques. International Journal of Computer Applications, (4), 139-149.

[7] Tang, P., Qiu, W., Huang, Z., Lian, H., & Liu, G. (2020). Detection of SQL injection based on artificial neural network. Knowledge-Based Systems, 105528. doi:10.1016/j.knosys.2020.105528

[8] Kamtuo, K., & Soomlek, C. (2016). Machine Learning for SQL injection prevention on server-side scripting. 2016 International Computer Science and Engineering Conference (ICSEC).

[9] Ross, K. (2018). SQL injection detection using machine learning techniques and multiple data sources.

[10] Qiang, W., & Zhongli, Z. (2011). Reinforcement learning model, algorithms and its application. 2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC).

[11] Tian, W., Yang, J.-F., Xu, J., & Si, G.-N. (2012). Attack Model Based Penetration Test for SQL Injection Vulnerability. 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops. doi:10.1109/compsacw.2012.108

[12] Ghanem, M. C., & Chen, T. M. (2019). Reinforcement learning for efficient network penetration testing. Information, 11(1), 6.

[13] Niculae, S., Dichiu, D., Yang, K., & Bäck, T. (2020). Automating penetration testing using reinforcement learning.

[14] Hu, Z., Beuran, R., & Tan, Y. (2020). Automated Penetration Testing Using Deep Reinforcement Learning. 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW).

[15] Sadeghian, A., Zamani, M., & Abdullah, S. M. (2013). A Taxonomy of SQL Injection Attacks. 2013 International Conference on Informatics and Creative Multimedia.

[16] Rai, A., Miraz, M. M. I., Das, D., Kaur, H., & Swati. (2021). SQL Injection: Classification and Prevention. 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM).

[17] Medhane, M. H. A. S. (2013). Efficient solution for SQL injection attack detection and prevention. International Journal of Soft Computing and Engineering (IJSCE), 3, 396-398.

[18] Halfond, W. G., Viegas, J., and Orso, A. A Classification of SQL-Injection Attacks and Countermeasures. In SSSE (2006).

[19] Wei, K., Muthuprasanna, M., & Suraj Kothari. (2006). Preventing SQL injection attacks in stored procedures. Australian Software Engineering Conference (ASWEC'06).

[20] Dr. Drew Hwang, "SQL Injection", 2022, https://hwang.cisdept.cpp.edu/swanew/text/SQL-Injection.htm

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ''BERT: Pre-training of deep bidirectional transformers for language understanding,'' 2018, arXiv:1810.04805.

[22] Shankar Das, S., Serra, E., Halappanavar, M., Pothen, A., & Al-Shaer, E. (2021). V2W-BERT: A Framework for Effective Hierarchical Multiclass Classification of Software Vulnerabilities. arXiv e-prints, arXiv-2102.

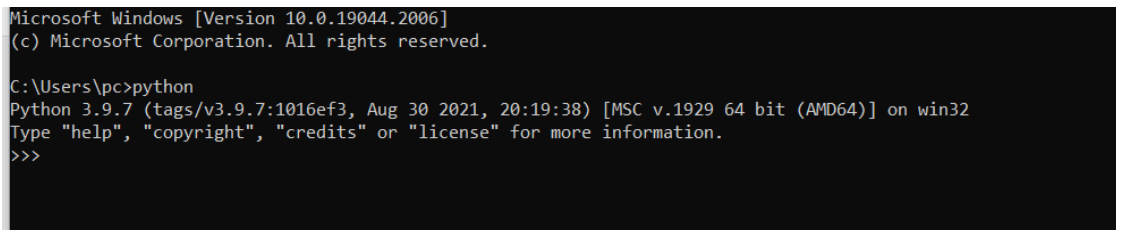
[23] "OWASP Top Ten", https://owasp.org/www-project-top-ten

Appendix

Appendix A
Python Download and Installation

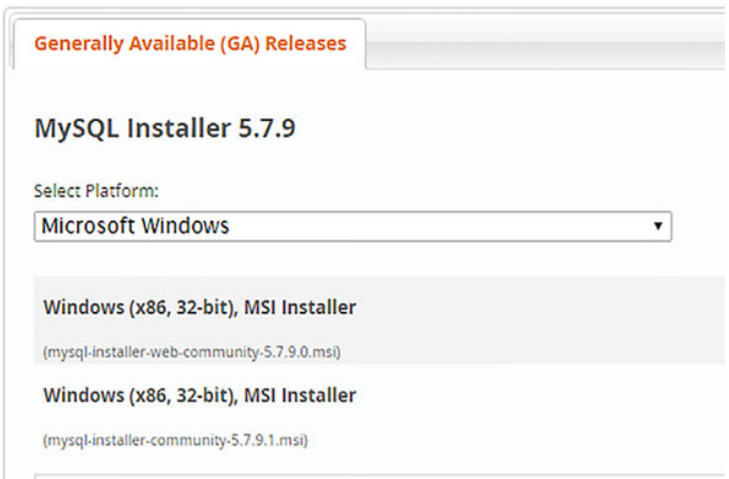
1. Visit the official website and go to https://www.python.org/downloads/. Click the Download button.

2. Once we click the download button, it might ask for a location to save the file. Select an appropriate location and then proceed towards the installation.
3. Double Click the downloaded .exe file and select the Add Python to PATH checkbox below to ensure it is automatically added to the Windows Environment variable. Else we have to do it later on manually. Once the box is checked, click on Install Now.
4. At the time of installation of python, the pop-up will show that the installation is in progress here.
5. Once the setup is complete, we will get a message like this. Click on the Close button to finish the installation of python.
6. Once Python is installed, go to the command prompt and type python , output like this should appear on successful installation.



Appendix B
MySQL Download and Installation

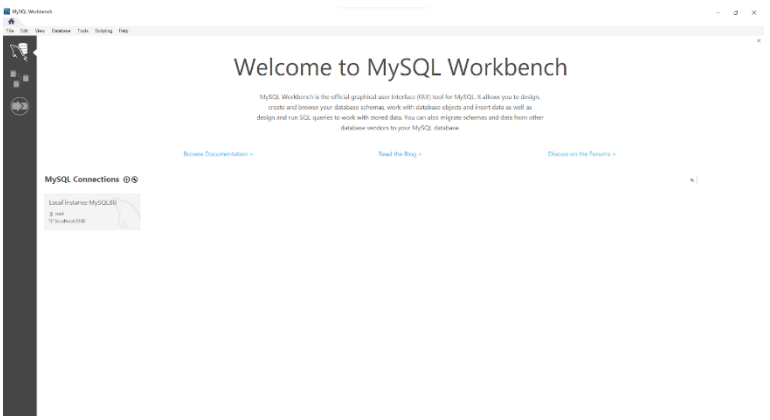
1. Download MySQL Installer from http://dev.mysql.com/downloads/installer/ and open it. Then select the suitable Setup Type you prefer.



2. Then the MySQL installation wizard's instructions will guide you through the setup

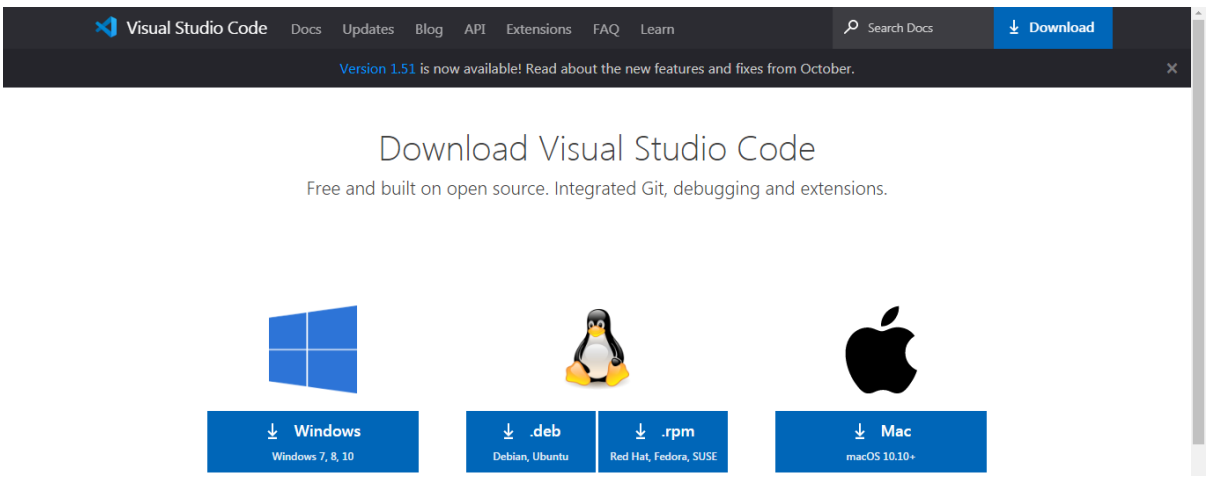
process. The installation completes. If you opted for a full install, by default MySQL server, MySQL workbench, and MySQL notifier will start automatically at computer startup.

3. The instance should be up and running, and you can connect to it using the MySQL workbench.

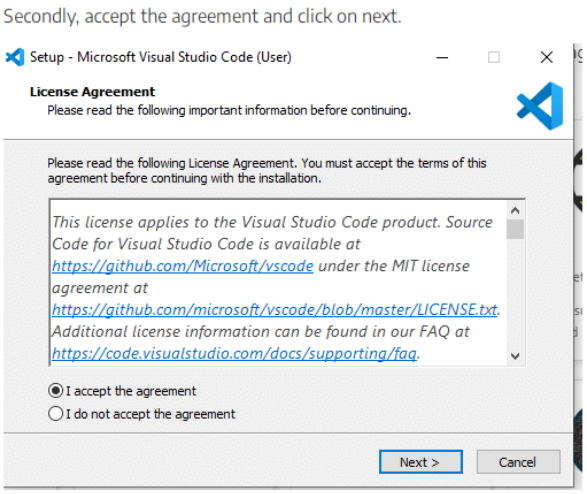


Appendix C
VS Code Download and Installation

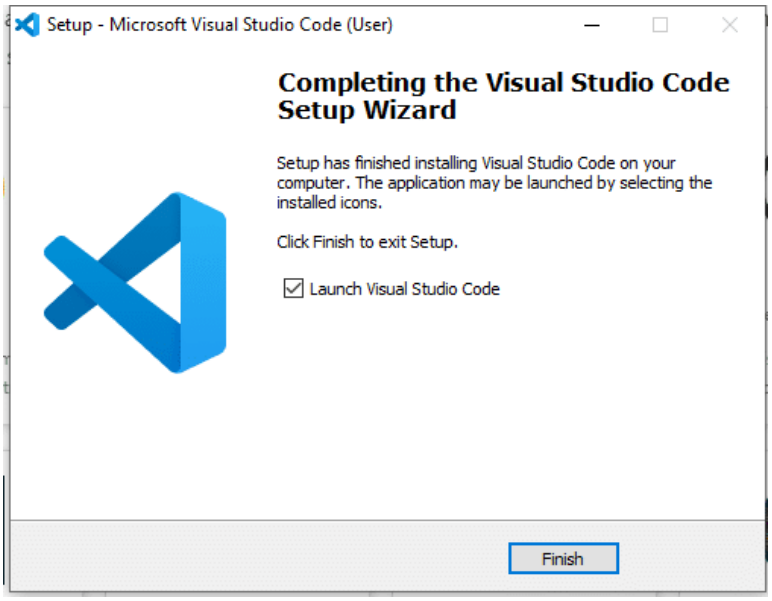
1. Download VS code from https://code.visualstudio.com/download
2. Download the Visual Studio Code installer for Windows. Once it is downloaded, run the installer (VSCodeUserSetup-{version}.exe). Then, run the file



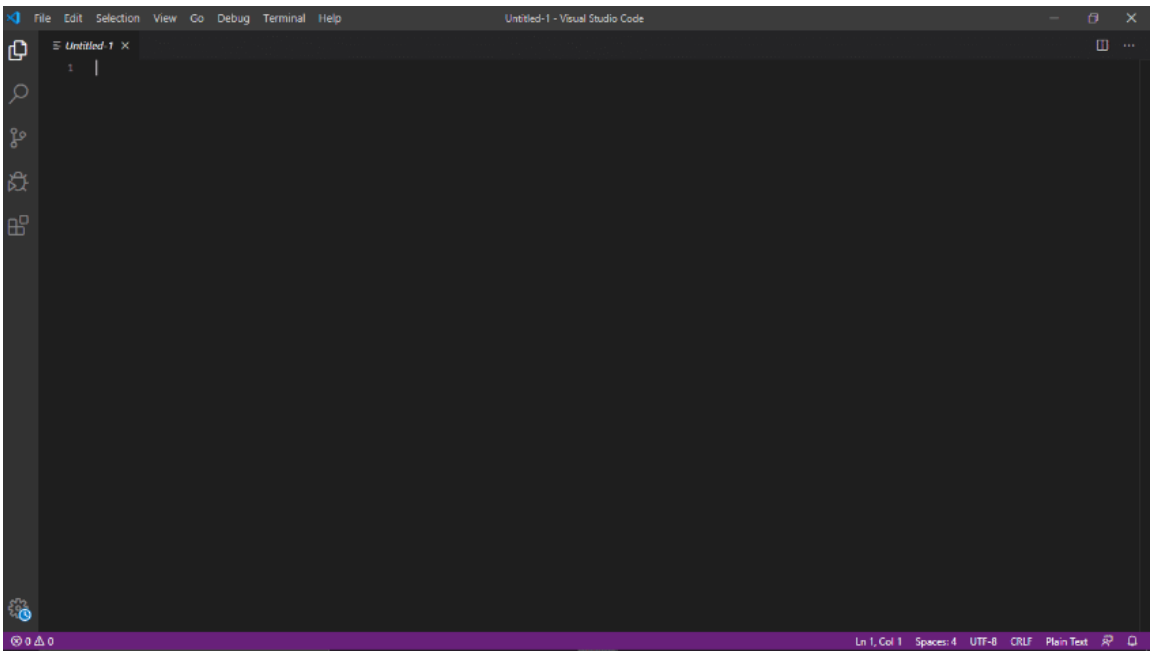
3. Accept the agreement and click “next.”



4. After accepting all the requests press the finish button. By default, VS Code installs under: "C:\users{username}\AppData\Local\Programs\Microsoft VS Code."



If the installation is successful, you will see the following



Appendix D
Packages and libraries installation

pip install requiremnts.txt

requirements.txt

```
flask
numpy
pandas
sklearn
scipy
beautifulsoup4
xgboost
mysql-connector-python
```