DETECTION OF SQL INJECTION USING REINFORCEMENT LEARNING

Submitted in partial fulfilments of the requirements of the degree of

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

University of Mumbai

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 fulfillment of the requirement for the award of the degree of Bachelor of Engineering in Computer Engineering.

_____ _____

Guide
Prof. Ramya R B

Project Coordinator
Prof. Rushikesh R. Nikam

_____ _____

Head of Department
Prof. Sachin H. Malave

Principal
Dr. Uttam D. Kolekar

Date:

A. P. SHAH INSTITUTE OF TECHNOLOGY, THANE

Project Report Approval for B.E.

This project report for Sem-VIII 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 the cause for disciplinary action by the Institute and can also invoke 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

SQL Injection is a major threat to web-based applications that rely on SQL databases to store and retrieve data. It allows attackers to inject malicious code into an SQL query, which can lead to data breaches and other security issues. Traditional machine learning (ML) methods have been used to detect SQL Injection attacks, but they are often limited by the lack of labelled data and the inability to detect zero-day attacks.

To address these issues, we propose a Q-Learning Reinforcement Learning approach for SQL Injection detection. This approach utilizes a combination of Q-Learning, Convolutional Neural Networks (CNNs), and the Apriori algorithm to detect SQL Injection attacks in SQL queries. The Q-Learning algorithm trains the model to make decisions about whether an SQL query is malicious or benign based on the rewards or penalties it receives during its interaction with the environment. The CNNs are used to analyze the SQL queries and identify potential SQL Injection attacks. The Apriori algorithm is used to identify frequent sequences of SQL queries that may indicate an SQL Injection attack.

We evaluate our approach on a dataset of SQL queries containing both benign and malicious queries. Our experimental results show that our approach achieves high accuracy and outperforms other state-of-the-art ML methods for SQL Injection detection. This approach can be applied to various web-based applications to prevent SQL Injection attacks, which are a major security threat to web-based applications that rely on SQL databases.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abbreviation

SQL	Structured Query Language
RDMS	Relational Database Management System
RL	Reinforcement Learning
ML	Machine Learning
SQLi	Structured Query Language Injection
SQLiA	Structured Query Language Injection Attacks
NB	Naïve Bayes
UML	Unified Modelling Language
CNN	Convolutional Neural Network
LR	Learning Rate
DF	Discount Factor

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].
    a. 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>
    b. 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>
    c. 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>
    d. 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

a. 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.

b. 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.

c. 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.

	Paper Name	**Authors**	**Research Finds**
[1]	Simulating SQL Injection Vulnerability Exploitation Using Q-Learning Reinforcement Learning Agents	Erdődi, L., Sommervoll, Å. Å., & Zennaro, F. M. (2021)	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.
[2]	Modelling Penetration Testing with Reinforcement Learning Using Capture-the-Flag Challenges	Zennaro, F. M., & Erdodi, L. (2020)	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.
[3]	SQL Injections and Reinforcement Learning: An Empirical Evaluation of the Role of Action Structure	Verme, M. D., Sommervoll, Å. Å., Erdődi, L., Totaro, S., & Zennaro, F. M. (2021, November)	This paper did comparison between the performance of simple structured agents and agents which were provided with a priori knowledge.

Table 2.1: Literature Survey Table

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

SQL Injection is a type of web-based attack that allows attackers to inject malicious SQL statements into an application's database, potentially gaining access to sensitive information or even taking over the entire system. Traditional methods of SQL Injection detection rely on supervised machine learning models or manual inspection of SQL queries, both of which have limitations in terms of accuracy and scalability. Therefore, there is a need for more advanced and robust techniques that can detect SQL Injection attacks in real-time and prevent data breaches in web-based applications. In this project, we propose to use a Q-Learning Reinforcement Learning approach with CNN apriori on a dataset to detect SQL Injection attacks. This approach has the potential to overcome the limitations of traditional methods and provide more accurate and scalable SQL Injection detection in web-based applications.

4.2Objectives

- Develop a comprehensive understanding of SQL Injection attacks and their impact on web-based applications.To predict the Health of Heart based on either Report data, Audio data or Image data of cardiogram.
- Review and analyze existing techniques for SQL Injection detection, including supervised machine learning models and manual inspection of SQL queries.

- Develop an understanding of the principles of reinforcement learning and its potential applications in SQL Injection detection.
- Implement a Q-Learning Reinforcement Learning approach for SQL Injection detection, using apriori on a dataset.
- Evaluate the performance of the proposed approach and compare it with existing techniques for SQL Injection detection, in terms of accuracy, scalability, and efficiency.
- Investigate the impact of different hyperparameters on the performance of the proposed approach, such as the learning rate, discount factor, and exploration rate.
- Identify and analyze the factors that affect the performance of the proposed approach, such as the complexity of the SQL queries, the size of the dataset, and the frequency of attacks.

4.3 Scope

The scope of the project that uses Q-Learning Reinforcement Learning approach to detect SQL Injection using CNN apriori is to develop an effective and efficient solution for detecting SQL Injection attacks in web-based applications. The project involves the development of a machine learning-based approach that can automatically detect SQL Injection attacks and prevent them from compromising sensitive data.

The project's scope includes a comprehensive review of existing techniques for SQL Injection detection, including supervised machine learning models and manual inspection of SQL queries. The project will then investigate the potential of reinforcement learning in detecting SQL Injection attacks and develop a Q-Learning Reinforcement Learning approach using CNN apriori on a dataset.

The developed approach will be evaluated in terms of accuracy, scalability, and efficiency, and compared with existing techniques for SQL Injection detection. The project also aims to investigate the impact of different hyperparameters on the performance of the proposed approach and identify the factors that affect its performance, such as the complexity of SQL queries, the size of the dataset, and the frequency of attacks.

The project's scope also includes the development of techniques for visualizing the results of the proposed approach and interpreting the decision-making process of the model. The proposed approach's robustness to different types of attacks and variations in the dataset will also be investigated.

Overall, the scope of the project is to develop an effective and efficient solution for detecting SQL Injection attacks in web-based applications using a machine learning-based approach. The project aims to contribute to the development of more accurate and scalable techniques for detecting SQL Injection attacks and ultimately help prevent data breaches and minimize the impact of attacks on web-based applications.

CHAPTER 5

Proposed System

5.1 Proposed System Overview

Develop a comprehensive understanding of SQL Injection attacks and their impact on web-based applications.

Review and analyze existing techniques for SQL Injection detection, including supervised machine learning models and manual inspection of SQL queries.

Develop an understanding of the principles of reinforcement learning and its potential applications in SQL Injection detection.

Implement a Q-Learning Reinforcement Learning approach for SQL Injection detection, using apriori on a dataset.

Evaluate the performance of the proposed approach and compare it with existing techniques for SQL Injection detection, in terms of accuracy, scalability, and efficiency.

Investigate the impact of different hyperparameters on the performance of the proposed approach, such as the learning rate, discount factor, and exploration rate.

Identify and analyze the factors that affect the performance of the proposed approach, such as the complexity of the SQL queries, the size of the dataset, and the frequency of attacks.

5.2 Design Details

5.2.1 Architecture Diagram

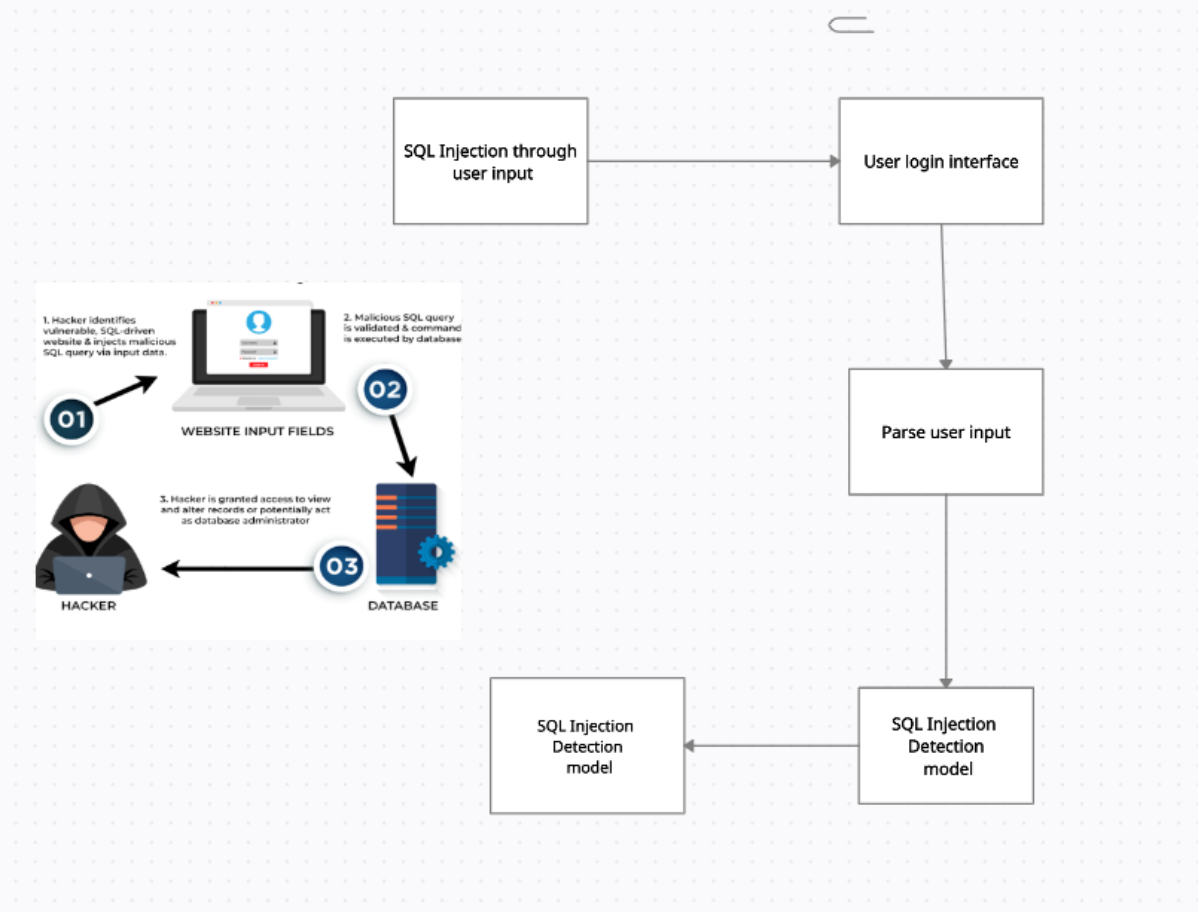


Fig 5.2.1: Architecture Diagram

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. 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.

5.2.2 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.

i. DFD Level 0

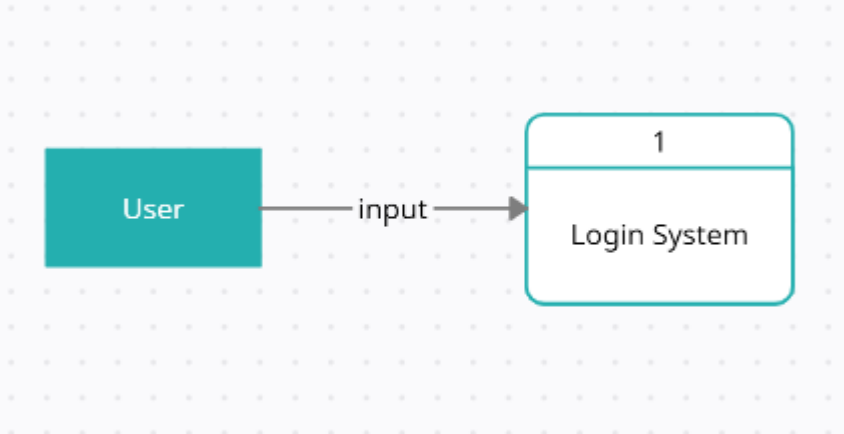


Fig 5.2.1.1: Data Flow Diagram Level 0

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.

ii. DFD Level 1

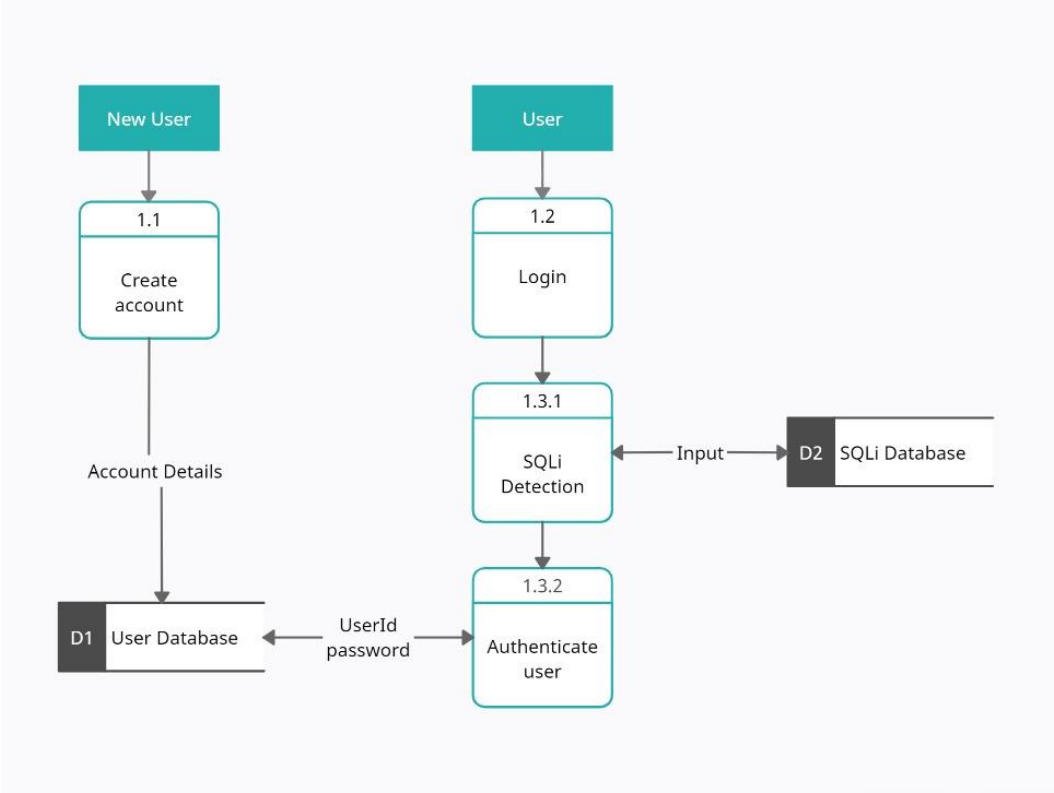


Fig 5.2.2.1: Data Flow Diagram Level 1

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.

5.2.3 Sequence Diagram

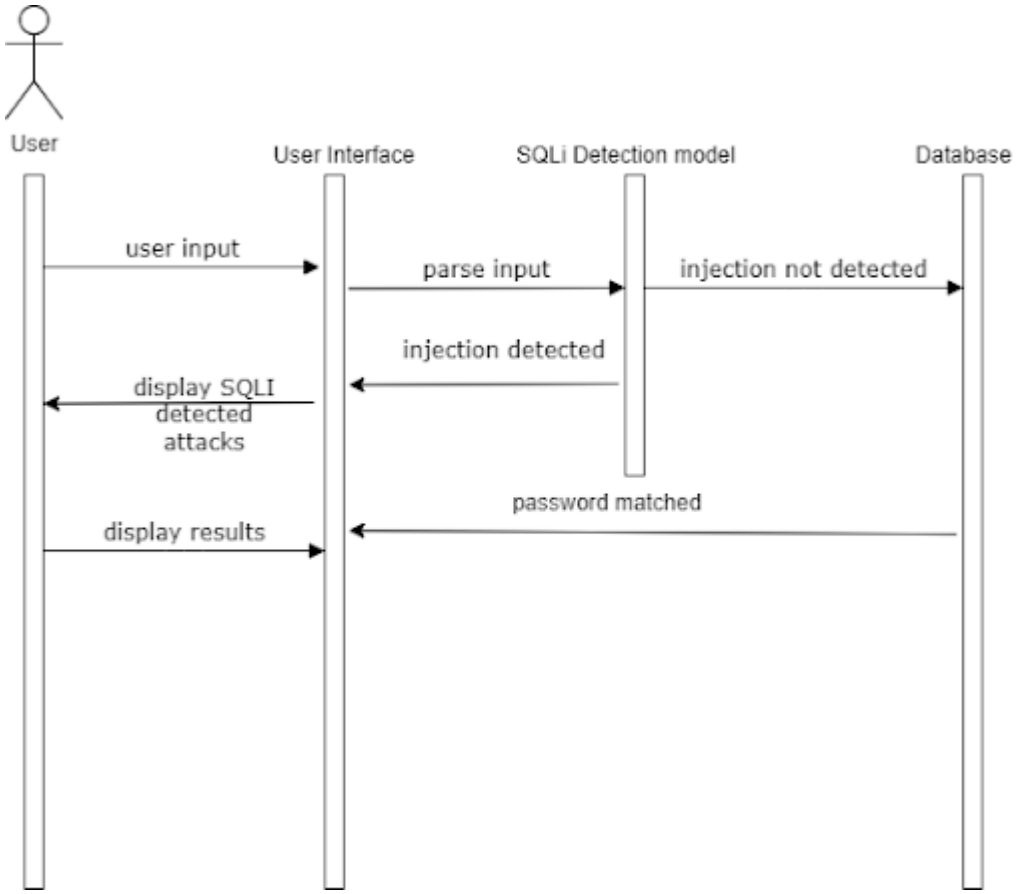


Fig 5.2.3: Sequence Diagram

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.

5.2.4 Activity Diagram

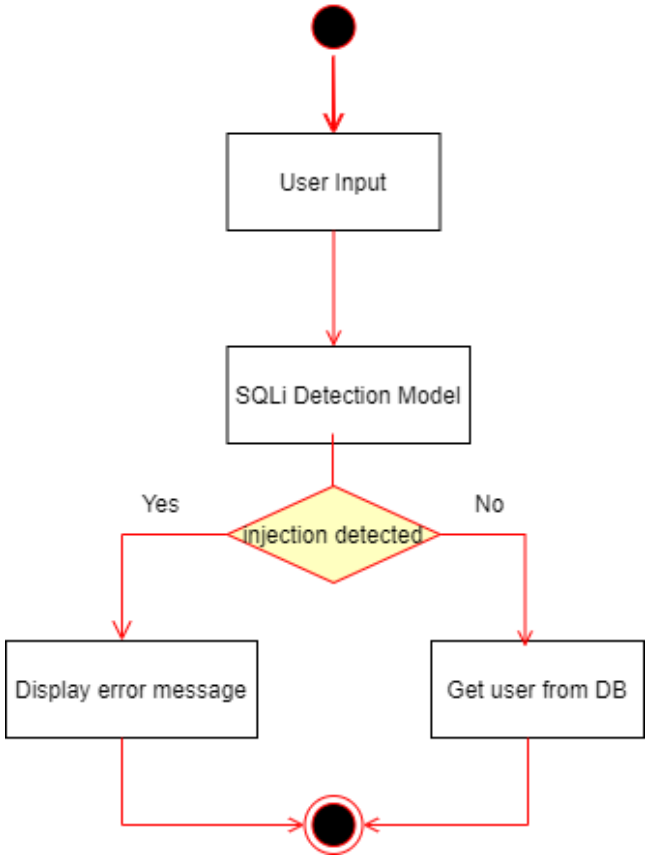


Fig 5.2.4: Activity Diagram

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.

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.

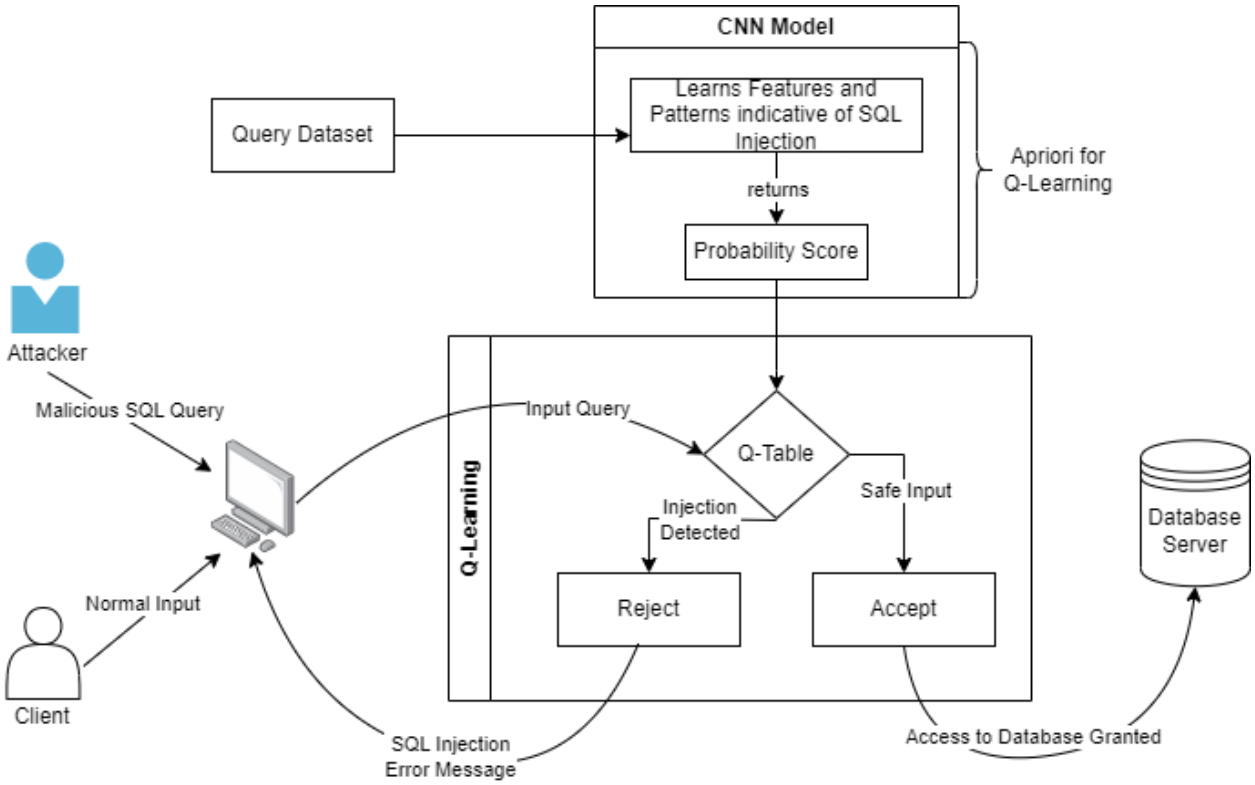


Fig 5.3: Detailed Architecture System

Fig. 5.3 depicts the system architecture where we provided the 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.

5.3.1 Working of Model

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 an 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.

5.3.2 Working 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.

5.3.3 Implementation

```
class QLearningBinaryClassifier:
    def __init__(self, num_features, learning_rate,
discount_factor):
        self.num_features = num_features
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.weights = np.zeros(num_features)

    def predict(self, state):
        q_value = np.dot(self.weights, state)
        action = 1 if q_value >= 0 else 0
        return action

    def update(self, state, action, reward, next_state):
        q_value = np.dot(self.weights, state)
        next_q_value = np.dot(self.weights, next_state)
        target = reward + self.discount_factor * next_q_value
        error = target - q_value
        self.weights += self.learning_rate * error * state
```

This class implements a binary classifier that uses Q-learning to learn the optimal action to take given a state. The __init__ method initializes the class with parameters such as the number of features in the input state, the learning rate, and the discount factor. The predict method takes in a state and returns an action (either 0 or 1) based on the dot product of the weights and the state. The update method updates the weights based on the reward received for taking the previous action and the predicted Q-value for the next state. The weights are updated using the learning rate and the error between the target and predicted Q-value.

```
class SQLInjectionAgent:
    def __init__(self):
        self.model    =    QLearningBinaryClassifier(num_features=5,
learning_rate = 0.1, discount_factor = 0.9)  # Define the RL model
(e.g., Q-learning, SARSA)

    def act(self, state):
        action = self.model.predict(state)
        return action
```

The SQLInjectionAgent class uses Q-learning binary classifier to make decisions related to detecting SQL injection. The class is initialized with a Q-learning binary classifier model and the act method uses the model to predict the best action to take given an input state.

```
# apriori
def is_injection(xi):
    return cnn_model.predict(xi.reshape(-1,1,4717)).flatten()

# Define the RL environment (QLearining model )
class SQLInjectionEnv(gym.Env):
    def __init__(self):
        self.state = None
        self.done = False
    def reset(self):
        self.done = False
    def step(self, state, action):
        query = state
        result = is_injection(query)
        if is_injection(query):
            reward = -1
            self.done = True
        elif result == 'error':
            reward = -0.5
        else:
            reward = 0.5
        return query, reward, self.done, {}
```

The above code includes a function called is_injection that predicts whether an input is an SQL injection or not using a pre-trained CNN model. It also defines a custom environment class called SQLInjectionEnv with reset and step methods to generate a random SQL query and execute it with an action. The reward is determined based on the query result and the done flag is set to True if the query is an SQL injection.

```
# Train the RL agent
env = SQLInjectionEnv()
agent = SQLInjectionAgent()
learning_rate = 0.1
discount_factor = 0.9
q_model  =  QLearningBinaryClassifier(num_features,  learning_rate,
discount_factor)
```

```
out = []

num_episodes = len(X_train)-1

for state in X_train:
    done = False
    while not done:
        action = q_model.predict(state)
        out.append(action)
        next_state, reward, done, _ = env.step(state,action) #
Observe the next state, reward, and done flag
        q_model.update(state, action, reward, next_state)
        state = next_state
```

This code trains a reinforcement learning (RL) agent using the SQLInjectionEnv environment and SQLInjectionAgent class. The RL algorithm used is Q-learning, implemented by the QLearningBinaryClassifier class.

The training process involves iterating over the training data (X_train) and for each state in the data, the agent takes an action using the q_model.predict method, which returns the predicted action based on the current state. The env.step method is called with the current state and predicted action, which returns the next state, reward, and done flag. The q_model.update method is then called to update the Q-values based on the current state, action, reward, and next state. The process continues until the done flag is set to True, indicating the end of an episode.

The out list keeps track of the actions taken by the agent during the training process. The learning_rate and discount_factor hyperparameters are set to 0.1 and 0.9, respectively. The number of episodes is set to the length of X_train minus 1.

5.3.4 User Interface

This is the user interface on visiting the website.

Fig 5.3.4.1: 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 5.3.4.2: Successful Login

Fig 5.3.4.3: Injection Detected

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

5.3.5 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 Forest, CNN and KNN as shown in Fig. 5.3.5.1. Random Forest and CNN model have been generating the most consistent accuracy and F1 scores.

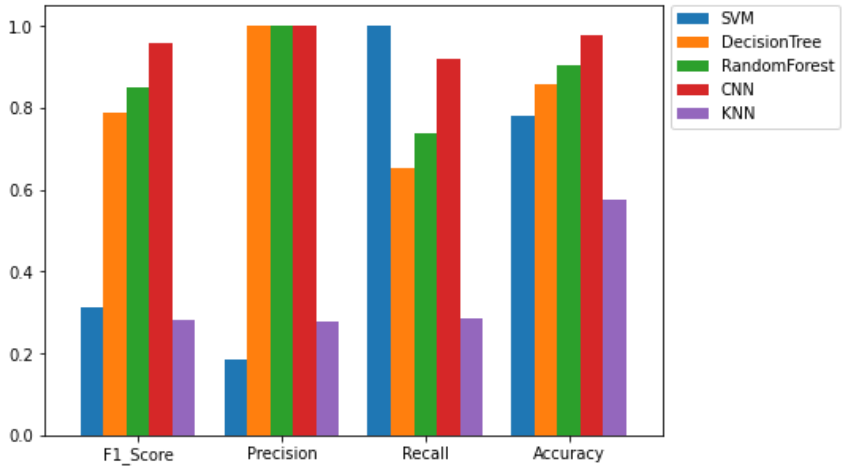


Fig 5.3.5.1: Comparison of the Models

Fig. 5.3.5.2 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.5.3.5.3.

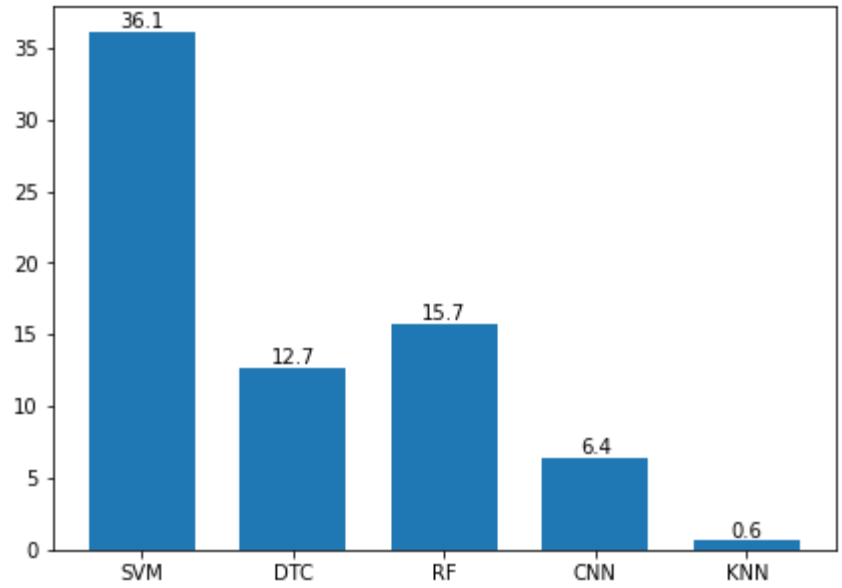


Fig 5.3.5.2: Compile Time for the Models

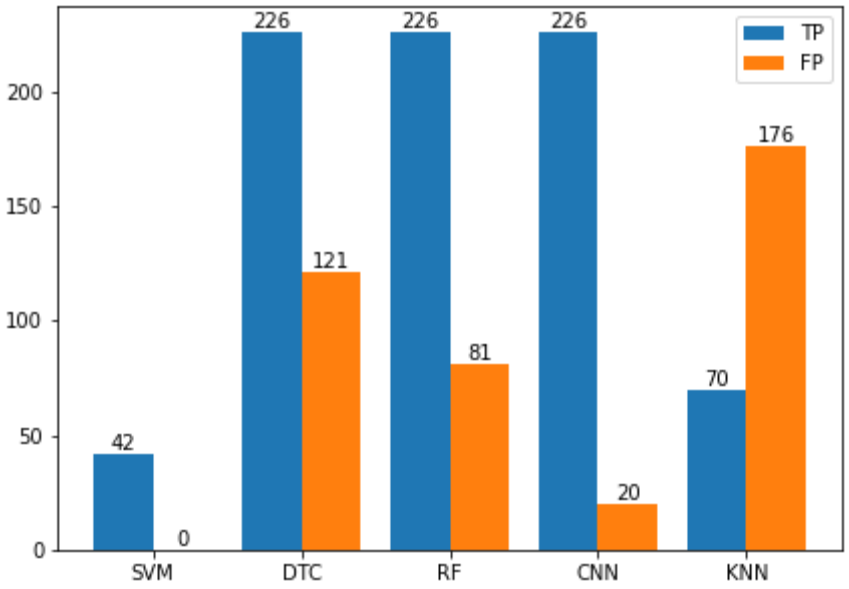


Fig 5.3.5.3: True Positive v/s False Positive

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.

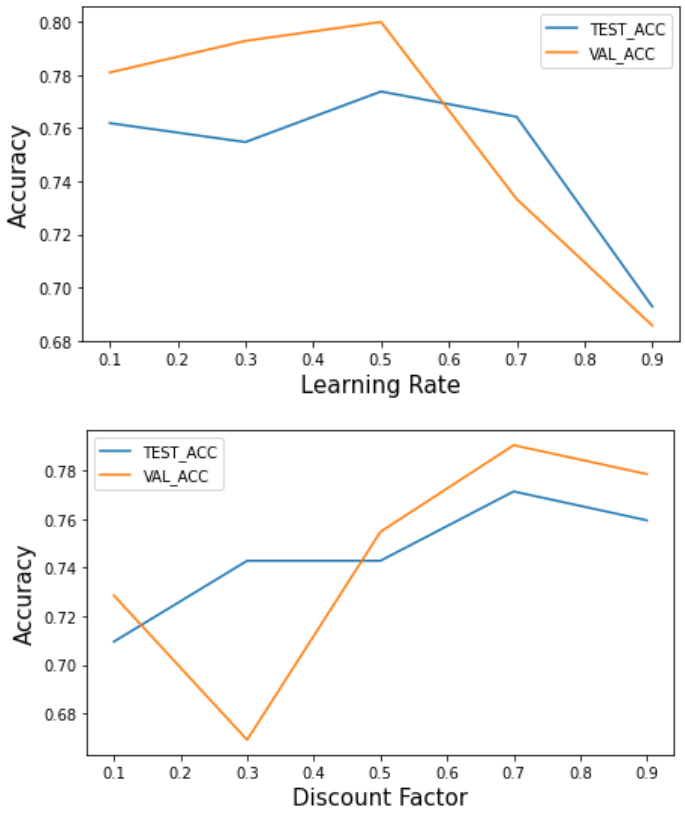


Fig 5.3.5.4: LR and DF trade-off with Accuracy

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.5.3.5.4. 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.

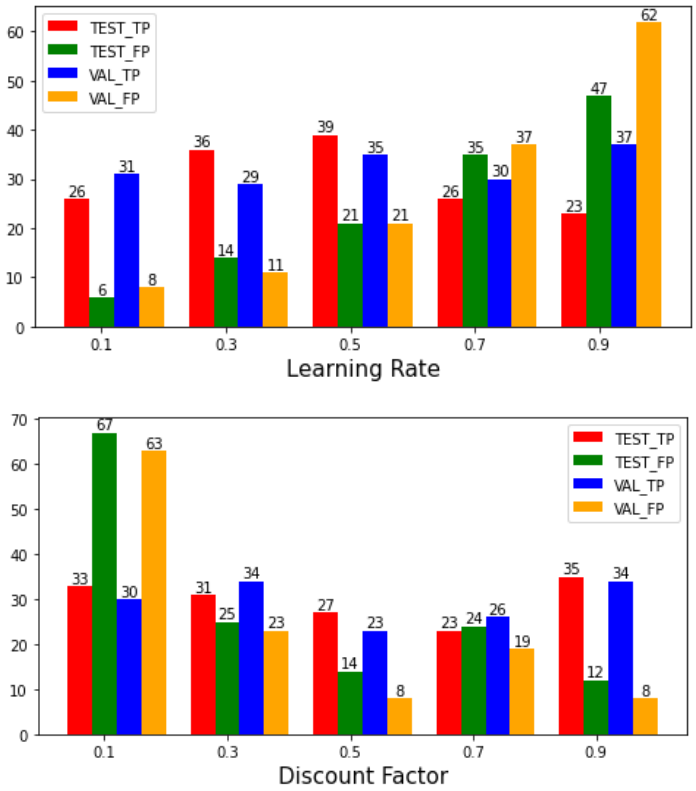


Fig 5.3.5.5: TP and FP for varying LR and DF

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.5.3.5.5.

CHAPTER 6

Limitation of Existing system

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. 5.3.5.4 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. 5.3.5.5.

CHAPTER 7

Experimental Setup

7.1 Software Requirements

1) Python: Python is a popular programming language used for machine learning and data science tasks. Python can be downloaded and installed from the official website.

2) TensorFlow: TensorFlow is an open-source machine learning library developed by Google. TensorFlow can be installed using pip or conda.

3) Keras: Keras is a high-level neural networks API written in Python that runs on top of TensorFlow. Keras can be installed using pip or conda.

4) scikit-learn: scikit-learn is a popular machine learning library for Python that provides a range of supervised and unsupervised learning algorithms. scikit-learn can be installed using pip or conda.

5) SQL databases: SQL databases such as MySQL, PostgreSQL, or SQLite might be required for storing and managing the dataset used in the project.

6) Visualization libraries: Visualization libraries such as Matplotlib, Seaborn, or Plotly might be required for visualizing the data and results.

7) Integrated Development Environment (IDE): An IDE such as PyCharm, Spyder, or Jupyter Notebook can be used for developing, testing, and debugging the code.

8) Git: Git is a version control system used for tracking changes in the code and collaborating with other team members. Git can be downloaded and installed from the official website.

9) HTML, CSS, and JavaScript: These are the core web development technologies used for creating the front-end user interface. HTML provides the structure of the web page, CSS provides the styling and layout, and JavaScript provides the interactivity and functionality.

10) Flask: Flask is a popular Python web framework used for developing web applications.

Flask can be used to create a server-side application that interacts with the machine learning model.

11) Visual Studio Code: Visual Studio Code is a popular code editor used for developing web applications and machine learning models. It provides a range of useful features such as code highlighting, auto-completion, and debugging.

12) TeamGantt: TeamGantt is a cloud-based project management tool that can be used to create and manage Gantt charts. It allows team members to collaborate and track the progress of tasks in real-time. TeamGantt also provides features such as resource allocation, time tracking, and task dependencies. It can be useful for managing complex projects involving multiple team members and tasks.

7.2 Hardware Requirements

1) Computer: A modern computer with a minimum of 8 GB RAM, multi-core processor, and at least 500 GB hard disk space is recommended. The computer should have a dedicated graphics card to accelerate the computation of deep learning models.

2) GPU: A GPU (Graphics Processing Unit) is highly recommended for training and evaluating deep learning models, especially for large datasets. NVIDIA GPUs such as GeForce RTX, Tesla, or Titan are widely used for deep learning tasks.

3) Storage: Adequate storage is essential for storing the large datasets used in the project. An external hard drive or a cloud-based storage solution such as Google Drive, Dropbox, or Amazon S3 can be used to store the data.

4) Network: A fast and reliable network connection is important for downloading large datasets and training deep learning models on cloud-based platforms such as Amazon Web Services, Google Cloud, or Microsoft Azure.

CHAPTER 8

Project Plan

Our project started in July 2022, and after finalizing the topic and objectives, we began working on it. Phase one covered project conception and design layout. In phase two, we designed our Q-Learning Model system, and in the final phase, we completed frontend implementation, and submitted a research paper along with our results.

Gantt Chart

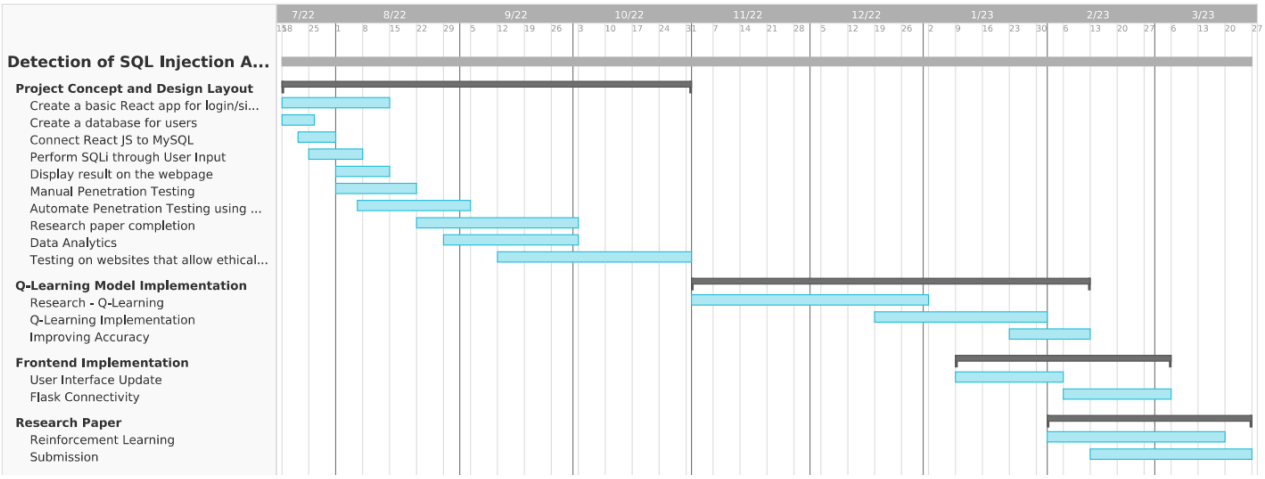


Fig 7.1: Gantt Chart

CHAPTER 9

Conclusion

In conclusion, being able to predict the status of heart health based on textual, audio & image data is a crucial step in preventing heart-related ailments. The heart disease prediction project is a complex and ambitious undertaking that involves the application of machine learning techniques to different types of data, including textual data, ECG, and audio data. The project aims to develop a comprehensive system that can accurately predict the risk of heart disease in patients based on various factors. The project has significant potential to improve the accuracy of heart disease diagnosis and treatment, leading to better health outcomes for patients.

One of the key strengths of the project is its interdisciplinary nature, involving experts from various fields such as computer science, data analysis, and healthcare. This interdisciplinary approach has led to the development of innovative solutions and techniques that can be applied to a wide range of healthcare problems beyond heart disease.

With the use of advanced technologies and machine learning algorithms, medical professionals can analyze the patterns and changes in the data to diagnose heart disease at an early stage. This can help individuals take preventive measures and improve their lifestyle to reduce the risk of heart disease. As technology continues to evolve, we can expect more accurate and efficient methods of predicting heart disease and ultimately improving the overall health of society.

CHAPTER 10

Future Scope

Here are some areas where the project can have a significant impact in the future:

Extension to other types of attacks: The proposed approach can be extended to detect and prevent other types of attacks, such as cross-site scripting (XSS) attacks and code injection attacks, in addition to SQL Injection attacks. This can be achieved by training the model on a dataset that includes examples of different types of attacks and developing appropriate reward functions to incentivize the model to detect and prevent them.

Integration with existing security systems: The proposed approach can be integrated with existing security systems in web-based applications, such as firewalls and intrusion detection systems (IDS), to provide an additional layer of protection against SQL Injection attacks. This can be achieved by developing APIs and plugins that can be easily integrated into existing systems and providing training and support to security professionals to use the approach effectively.

Application in other domains: The proposed approach can be applied to other domains beyond web-based applications, such as IoT devices and cloud computing systems, to detect and prevent attacks that exploit vulnerabilities in the software and hardware. This can be achieved by developing appropriate reward functions and training the model on datasets that are specific to each domain.

Optimization of hyperparameters: The proposed approach can be further optimized by experimenting with different hyperparameters, such as the learning rate, discount factor, and exploration rate, to achieve better performance in terms of accuracy, scalability, and efficiency. This can be achieved by conducting systematic experiments and analyzing the impact of each hyperparameter on the performance of the model.

Integration with explainable AI techniques: The proposed approach can be integrated with explainable AI techniques, such as decision trees and rule-based systems, to provide a more interpretable and transparent decision-making process. This can help security professionals better understand the model's decisions and take appropriate actions to prevent attacks.

Development of a comprehensive toolkit: The proposed approach can be developed into a comprehensive toolkit that includes a set of tools and libraries for training and deploying the model, visualizing the results, and monitoring the system's performance. This can help security professionals easily adopt and use the approach in their organizations and stay up-to-date with the latest developments in the field.

References

[1] “OWASP Top Ten”, https://owasp.org/www-project-top-ten.

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

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

[4] 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).

[5] 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.

[6] 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.

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

[9] 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).

[10] 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.

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

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

[13] 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).

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

[15] 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.

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

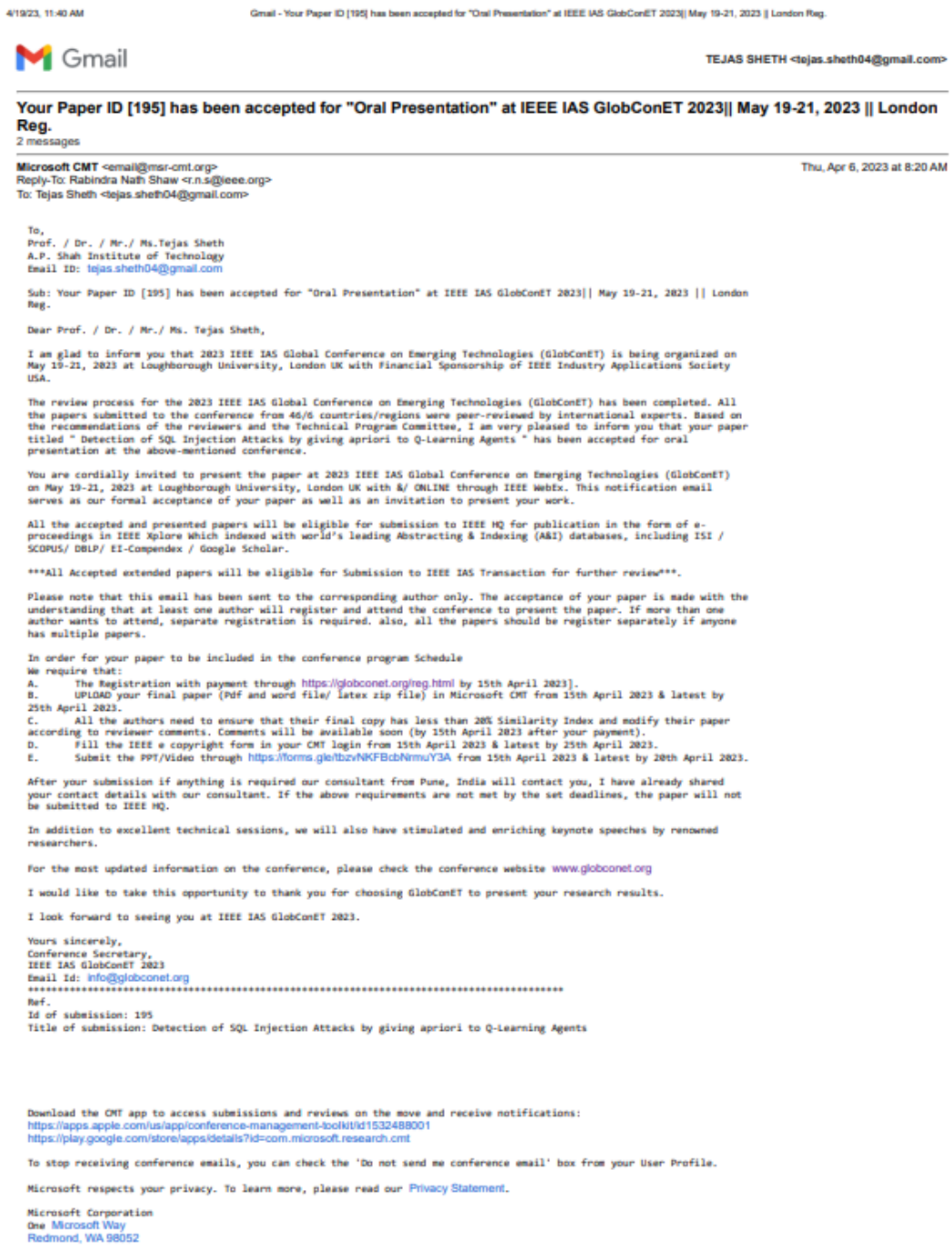
Publications

Paper 1: Detection of SQL Injection Attacks by giving apriori to Q-Learning Agents

Conferences:

[1] 2023 IEEE IAS Global Conference on Emerging Technologies (GlobConET 2023)

Status: **Accepted**



4/19/23, 11:40 AM Gmail - Your Paper ID [195] has been accepted for "Oral Presentation" at IEEE IAS GlobConET 2023|| May 19-21, 2023 || London Reg.

Gmail TEJAS SHETH <tejas.sheth04@gmail.com>

Your Paper ID [195] has been accepted for "Oral Presentation" at IEEE IAS GlobConET 2023|| May 19-21, 2023 || London Reg.

2 messages

Microsoft CMT <email@msr-cmt.org> Thu, Apr 6, 2023 at 8:20 AM
Reply-To: Rabindra Nath Shaw <r.n.s@ieee.org>
To: Tejas Sheth <tejas.sheth04@gmail.com>

To,
Prof. / Dr. / Mr./ Ms.Tejas Sheth
A.P. Shah Institute of Technology
Email ID: tejas.sheth04@gmail.com

Sub: Your Paper ID [195] has been accepted for "Oral Presentation" at IEEE IAS GlobConET 2023|| May 19-21, 2023 || London Reg.

Dear Prof. / Dr. / Mr./ Ms. Tejas Sheth,

I am glad to inform you that 2023 IEEE IAS Global Conference on Emerging Technologies (GlobConET) is being organized on May 19-21, 2023 at Loughborough University, London UK with Financial Sponsorship of IEEE Industry Applications Society USA.

The review process for the 2023 IEEE IAS Global Conference on Emerging Technologies (GlobConET) has been completed. All the papers submitted to the conference from 46/6 countries/regions were peer-reviewed by international experts. Based on the recommendations of the reviewers and the Technical Program Committee, I am very pleased to inform you that your paper titled " Detection of SQL Injection Attacks by giving apriori to Q-Learning Agents " has been accepted for oral presentation at the above-mentioned conference.

You are cordially invited to present the paper at 2023 IEEE IAS Global Conference on Emerging Technologies (GlobConET) on May 19-21, 2023 at Loughborough University, London UK with &/ ONLINE through IEEE WebEx. This notification email serves as our formal acceptance of your paper as well as an invitation to present your work.

All the accepted and presented papers will be eligible for submission to IEEE HQ for publication in the form of e-proceedings in IEEE Xplore Which indexed with world's leading Abstracting & Indexing (A&I) databases, including ISI / SCOPUS/ DBLP/ EI-Compendex / Google Scholar.

All Accepted extended papers will be eligible for Submission to IEEE IAS Transaction for further review.

Please note that this email has been sent to the corresponding author only. The acceptance of your paper is made with the understanding that at least one author will register and attend the conference to present the paper. If more than one author wants to attend, separate registration is required. also, all the papers should be register separately if anyone has multiple papers.

In order for your paper to be included in the conference program Schedule
We require that:
A. The Registration with payment through https://globconet.org/reg.html by 15th April 2023].
B. UPLOAD your final paper (Pdf and word file/ latex zip file) in Microsoft CMT from 15th April 2023 & latest by 25th April 2023.
C. All the authors need to ensure that their final copy has less than 20% Similarity Index and modify their paper according to reviewer comments. Comments will be available soon (by 15th April 2023 after your payment).
D. Fill the IEEE e copyright form in your CMT login from 15th April 2023 & latest by 25th April 2023.
E. Submit the PPT/Video through https://forms.gle/tbzvNKFBcbNrmuY3A from 15th April 2023 & latest by 20th April 2023.

After your submission if anything is required our consultant from Pune, India will contact you, I have already shared your contact details with our consultant. If the above requirements are not met by the set deadlines, the paper will not be submitted to IEEE HQ.

In addition to excellent technical sessions, we will also have stimulated and enriching keynote speeches by renowned researchers.

For the most updated information on the conference, please check the conference website www.globconet.org

I would like to take this opportunity to thank you for choosing GlobConET to present your research results.

I look forward to seeing you at IEEE IAS GlobConET 2023.

Yours sincerely,
Conference Secretary,
IEEE IAS GlobConET 2023
Email Id: info@globconet.org
****
Ref.
Id of submission: 195
Title of submission: Detection of SQL Injection Attacks by giving apriori to Q-Learning Agents

Download the CMT app to access submissions and reviews on the move and receive notifications:
https://apps.apple.com/us/app/conference-management-toolkit/id1532488001
https://play.google.com/store/apps/details?id=com.microsoft.research.cmt

To stop receiving conference emails, you can check the 'Do not send me conference email' box from your User Profile.

Microsoft respects your privacy. To learn more, please read our Privacy Statement.

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Paper 2: Comparative Study on Detection of SQL Injection Vulnerabilities using various Machine Learning Algorithms

Conferences:

[1] IEEEEC SMARTGEN Conference

Status: **Submitted**

Detection of SQL Injection Attacks by giving apriori to Q-Learning Agents

Tejas Sheth
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
tejas.sheth04@gmail.com

Janhavi Anap
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
anapjanhavi@gmail.com

Het Patel
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
hetpokar@gmail.com

Nidhi Singh
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
singhnidhik2002@gmail.com

Prof. Ramya R B
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
ramyarb@apsit.edu.in

***Abstract*—Software developers may have created the SQL Injection vulnerability accidentally, or a hacker may have purposefully used it to target vulnerable 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 study, we examine the outcomes of algorithms for reinforcement learning, such as Q-Learning, using a dataset made up of probable 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**

I. Introduction

Cybersecurity is the method of defending systems and programs from digital strikes. These strikes are usually designed for retrieving, manipulating or destroying sensitive data or interrupting standard business approaches. Cybersecurity aims to reduce the risk of cyber-attacks and to protect against the illegal manipulation of networks and systems. Implementing robust and reliable measures is challenging due to the ever-evolving innovative attackers.

Structured Query Language Injection Attacks (SQLiA) are one of the most dangerous security flaws that are frequently employed by hackers, according to the Top 10 report from the Open Web Application Security Project (OWASP) for 2022 [1]. By inserting malicious data into a database request, hackers can get access to a database without authorization by using SQL Injection (SQLi).

Data held in relational databases can be created, retrieved, updated, and deleted using the computer programming language SQL. All relational database management systems (RDBMS), including MySQL, employ SQL, a standard database language, extensively. SQL has the advantage of allowing users to access, define, describe, and alter data. Additionally, it permits users to make databases, drop and create tables, make views, store procedures, and functions, as well as control permissions for tables, views, and procedures. SQL enables the use of pre-compilers, libraries, and modules with other languages.

The main reason why SQLi keeps traction is due to improper and insecure development architecture. Making use of legacy software introduces security vulnerabilities that might not be detected with modern software. Using authorized and the latest versions of the software are critical to avoiding or minimizing potential security exploits, including SQLiA.

SQLiA mainly occurs through the concatenation of a malicious SQL command at the end of a given input query on the user front end. In recent times, there has been an advancement in the number of dynamic websites, that rely on relational databases to store and manipulate large amounts of data to provide a rich user experience. The SELECT command in conjunction with the WHERE clause is used to retrieve data, which is one of the most popular SQL actions.

A few of the common SQLiA types are Piggy-Backed, Stored Procedures, Union Query, and Alternative Encoding according to Halfond, W. G. et. al. [2] and Wei, K. et. al. [3]. There are some advanced SQLiA types like Blind SQLi where the attackers devise a strategy to circumvent the lack of error notifications, Fast Flux SQLi where the DNS method is used to conceal phishing and malware distribution sites behind a constantly changing network of compromised servers, Compounded SQLi is when the attacker combines two or more attacks that target the website and has improbable repercussions.

The detection of these attack types becomes a very important task keeping in mind the potential data breaches that could happen. The detection techniques include keeping an eye on database issues and reviewing server logs. The most popular method for identifying SQL Injection threats is pattern matching, which combines static analysis and real-time monitoring.

Another approach to detecting SQLiA is by making use of various Machine Learning techniques like supervised and unsupervised, Reinforcement Learning. There are many supervised and unsupervised approaches towards detecting SQLiA.

In this paper, we have used the Reinforcement Learning method towards SQLiA detection. Here, in the training phase, we provide apriori to the Q-Learning model. The training dataset contains both malicious and non-malicious queries. Providing efficient apriori is the most crucial factor in order to obtain higher efficiency. The advantage of Reinforcement Learning techniques over others is that it allows us to explore a variety of SQL queries and the accuracy keeps increasing with the increase in the use of the model. Also, owing to Reinforcement Learning methodologies, the accuracy value of SQL detection is raised and the false positive rate drops.

II. Literature Survey

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 [4]. 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.

Erdődi, L. et. al. simplified the dynamics of SQLi vulnerabilities by projecting the problem as capture-the-flag security and implementing it as an RL problem [5]. Assuming the vulnerability has been recognized, 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 Q-Learning algorithm can successfully develop an effective strategy to resolve the SQLi problem. Through pure trial and error, a tabular Q-Learning algorithm can uncover an effective strategy and achieve performance levels close to the theoretical optimum. However, while the use of a table to store the Q-value function enables a detailed analysis of the agent's learning dynamics, this approach suffers from poor scalability, limiting its practical applicability. Thus, in the second simulation, they sacrificed interpretability to work around the issue of scalability. In the first simulation, a deep Q-Learning agent was used to tackle the issue. It was able to learn an effective approach for the SQLi problem and also offer a solution to the space restrictions caused by the instantiation of an explicit Q-Table.

Verme, M. D. et. al. contemplated the problem of exploiting SQLi flaws and passing it off as a capture-In a "the-flag" scenario, an attacker may fill out a form with strings to obtain a flag token that represents personal data. [6]. The attacker was modelled as an RL agent that maintains interaction with the server to learn an optimal policy leading to the attacker taking advantage of it. The authors did a comparison between two types of agents, one was a simple structured agent that relied on significant apriori knowledge and used high-level actions while the other was a structureless agent that had limited apriori knowledge and generated SQL statements. The comparison demonstrated the viability of creating agents with fewer ad hoc modelling requirements.

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. Using Artificial Neural Network (ANN) models, the URL is categorised as harmful or non-malicious. The method and algorithm used here are multi-layer perceptrons (MLP) and LSTM, both 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. While using the LSTM model for recognition purposes in this approach, there were some drawbacks. These include reduced accuracy, poor model recognition, and increased processing time, which can be problematic for the overall performance of the system.

Niculae, S. et al. measured the performance of multiple fixed-strategy and learning-based agents [8]. 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. suggest an automated penetration testing framework, based on deep learning techniques, particularly Deep Q-Learning networks (DQN) [9]. 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.

RL is fairly successful in tackling and solving games, penetration testing, when distilled as a capture-the-flag (CTF), can be expressed as a game. When it comes to penetration testing, a machine may be limited to learning by trial and error, whereas a human hacker can utilize other methods such as knowledge from alternative sources, deduction, hypothesis testing, and social engineering. Although an RL agent may learn the structure completely without using any models, this may be computationally difficult. Thus according to Zennaro, F. M., & Erdodi, L. injecting some form of elementary apriori knowledge about the structure of the problem may simplify the learning problem [10]. Some basic forms of apriori knowledge are lazy loading, state aggregation and imitation learning, which makes the RL agent much more efficient. 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. The common RL interface

specified in the OpenAI gym library was used to implement each simulation.

Ghanem M. C., & Chen T. M. propose and evaluates an AI-based pen-testing system which makes use of RL to learn and replicate standard and complicated pen-testing activities [11]. The scope is limited to network infrastructure PT planning and not the entire practice. Moreover, the authors tackle the tricky problem of expertise capturing by allowing the learning module to store and reuse PT policies in a more efficient way.

John, A. contemplated methods incorporating the best qualities of parse tree validation and code conversion techniques [12]. The algorithm parses the user input and checks for vulnerability, and if found, it applies code conversion over that input. Results show few drawbacks of code conversion as applying it to every user input is labour-intensive as well as the database increases. The parse tree validation technique could raise a false alarm if a legitimate user is having blank space in their 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 [13]. The considered model works by sending different particularly 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. Typically, software developers employ string concatenation to dynamically build SQL statements. However, this approach can be prone to errors and may require manual intervention. The proposed method allows for the creation of multiple queries tailored to meet the different conditions specified by users. This approach eliminates the need for manual intervention and mitigates the risks associated with error-prone coding. The model showed consistency in attack detection and elimination at an average of 82% for all types of attacks.

By fusing a malicious SQL query with the input parameters, SQL Injection introduces a malicious SQL query into a web application. Sadeghian, A. et. al. illustrate how injection attacks are categorized, including tautology inquiries, illegal or illogical questions, union queries, piggy-backed queries, stored procedures, inference, and alternative encoding. [14]. Security researchers have categorized the solutions for SQLi into three major categories: 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.

Medhane and M. H. A. S. based their approach on grammar to determine the injection vulnerabilities during software development and SQLi attack based on web applications [15]. By employing SQL queries, the attackers were able to execute their assault, causing a modification in the program's behaviour as the SQL queries were modified, thus disrupting its intended function.

Reinforcement Learning (RL) is known to obtain knowledge by trial-and-error method and continuous interaction with a dynamic environment. It is characterized by self-improving and online learning, making it one of the intelligent agents (IA) core technologies. In reinforcement learning (RL), the signal provided by the environment serves as a form of evaluation for the quality of the actions taken by the AI. However, this signal does not provide instructions on 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 [16]. where the IA observes the surroundings and decides how to act to continuously engage with the surroundings to maximise reward value. The ultimate goal of RL is to learn an action strategy. At the core of reinforcement learning technology lies the idea that a system's actions which yield positive rewards from the environment will reinforce its tendency to produce similar actions in the future, thereby creating a positive feedback loop. Conversely, actions that do not produce a positive reward will weaken the system's tendency to generate similar actions. 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, like the Q-Learning algorithm.

III. Reinforcement Learning

An agent learns to make decisions by interacting with the environment via a machine learning technique called reinforcement learning. Here there are unlabeled data, so the agent learns through experience. To maximise a reward signal, the agent engages with the surroundings. The agent learns the concept of a reward-based mechanism using the hit-and-trial method. It is employed in a variety of applications, including gaming, robotics, and decision-making.

A. 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**- Reward is a measurement metric that the environment sends back to the agent. **Policy**- strategy applied by an agent for the next action based on the current state. **Value**-Value is the anticipated long-term return at a discount in comparison to the short-term benefit. **Q-Value**-Similar to the value, Q-Value also requires a parameter called current action. It is the anticipated overall benefit of acting in a certain state and abiding by a certain set of rules.

B. Reinforcement Learning Algorithms

1) Q-Learning: A model-free reinforcement learning algorithm is called Q-Learning. Additionally, it has a reputation for being able to handle non-stationary situations where the

underlying distribution of the data may shift over time and capture complicated relationships among input features. 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.

2) State Action Reward State action (SARSA): The main distinction between SARSA algorithms and Q-Learning is that SARSA algorithms do not require the greatest reward for the upcoming state in order to update 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. The agent employs SARSA during training to develop a policy that maximises the projected 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.

The SARSA algorithm is an on-policy alternative to Q-learning, 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 difficult and time-consuming task, especially when dealing with large datasets.

3) Deep Q-Learning: Deep neural networks and reinforcement learning are combined in this system, which allows for more complex representations of the state and action spaces. A neural network is used in the Q-learning variant known as "Deep Q-Learning" (DQL) 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. A new SQL query is seen by the agent, which then runs it through the neural network to determine the Q-values for each action and chooses the one with the greatest 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 effectively by approximating the Q-function using a neural network. 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.

C. Implementation of Q-Learning

1) Algorithm: The Q-learning algorithm constructs a table of Q-values that indicate the anticipated rewards for performing a certain action in a specific condition. The expected reward that can be attained by doing the action in the state in question and then implementing the best course of action is represented by the Q-value for that state-action pair. The algorithm starts by initializing the Q-table with arbitrary values, and then iteratively updates the Q-values based on the rewards received during each trial. An action (a) is taken based on the current state and Q-table. The Q-learning algorithm keeps going through iterations until the Q-values reach their ideal values. By choosing the action for each state that has the highest Q-value, the best policy can be found.

2) Integrating Apriori with Q-Learning: 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 the 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.

To integrate apriori in the Q-Learning model,

- Define the Q-learning environment model and its functions of it like reset and step. Also, define the Q-learning agent model and its function like act.
- The step function will take the query as input and with the help of the apriori of the CNN model it will determine the reward.
- The act function will make the prediction based on the policy the Q-Learning model has created.

3) Updating of Q-Table: 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 as follows:

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

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.

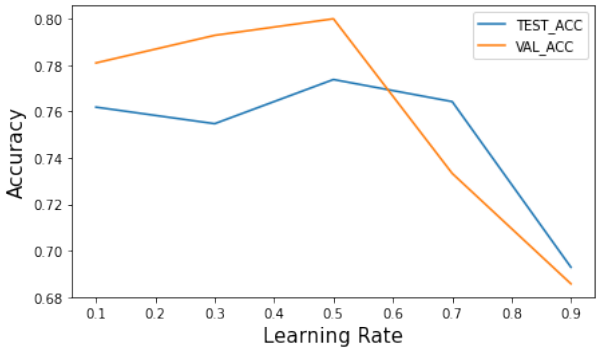


Fig. 1. Effect of Learning Rate on Accuracy

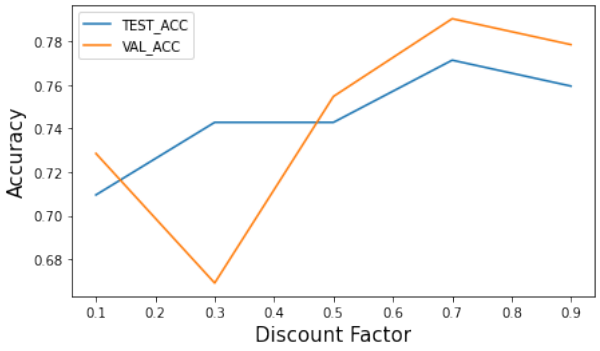


Fig. 2. Effect of Discount Factor on Accuracy

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 rewards.

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

IV. Experimental Outcomes

The Learning Rate (L.R.) 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. 1. 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 the optimal policy since it takes longer for the agent to learn from its experiences.

The Discount Factor (D.F.) 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 as seen in Fig. 2. A low discount factor (close to 0) means that the agent only cares about immediate rewards and does not consider future rewards.

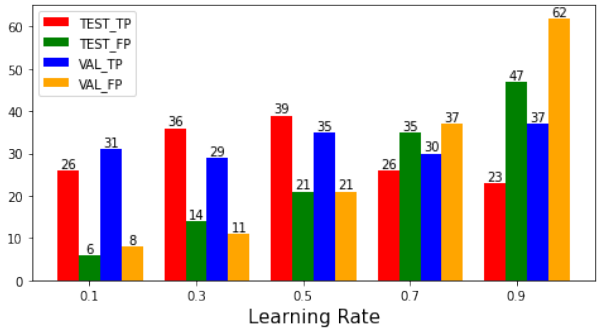


Fig. 3. Effect of Learning Rate on Testing and Validation Predictions

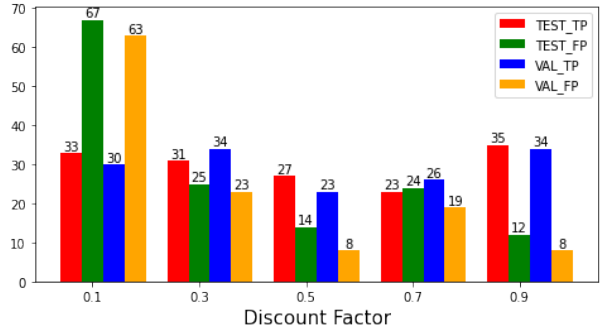


Fig. 4. Effect of Discount Factor on Testing and Validation Predictions

We have proved that a higher learning rate does not necessarily result in better accuracy for the testing and validation data. While the accuracy for testing and validation data was relatively consistent across the varying learning rates, the true positive and false positive rates varied significantly. For example, a learning rate of 0.1 had a lower true positive rate and a lower false positive rate than a learning rate of 0.9 for the testing data, while the opposite was true for the validation data. Exploring different combinations of learning rates and discount factors yields better performance.

Also, the proposed system resulted in significant accuracy for both testing and validation datasets. The highest accuracy achieved for testing data was 0.7714, while for validation data it was 0.7905, both achieved with a discount factor of 0.7.

V. 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. 1 and Fig. 2 where the accuracy for validation data is greater than the accuracy for testing data in an ideal condition. This increases the accuracy from 66% to around 70% for learning from training data. 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%. Fig. 3 and Fig. 4 visualize the values of True Positives and False Positives after learning from testing and then validation data with the varying values of Learning Rate and Discount Factor respectively. 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.

References

[1] “OWASP Top Ten”, https://owasp.org/www-project-top-ten

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

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

[4] 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).

[5] 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.

[6] 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.

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

[9] 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).

[10] 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.

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

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

[13] 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).

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

[15] 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.

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

Comparative Study on Detection of SQL Injection Vulnerabilities using various Machine Learning Algorithms

Tejas Sheth
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
tejas.sheth04@gmail.com

Janhavi Anap
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
anapjanhavi@gmail.com

Het Patel
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
hetpokar@gmail.com

Nidhi Singh
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
singhnidhik2002@gmail.com

Prof. Ramya R B
Department of Computer Engineering
A. P. Shah Institute of Technology
Thane (M.H.), India, 400615
ramyarb@apsit.edu.in

***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 warrant 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 and compare the results of different Machine Learning algorithms on a dataset consisting of potential SQL Injection queries. We intend to provide a Machine Learning solution to minimise the potential threat posed by SQL Injection, and develop a Reinforcement Learning algorithm to learn to detect a SQL attack and prevent any unforeseen mishap.**

***Keywords*—SQL injection, SQL detection, SQL prevention, SQLIA, Information security, Q-learning, Reinforcement learning, Vulnerability detection, Machine learning, Capture the flag**

I. Introduction

Cybersecurity is the method of defending systems and programs from digital strikes. These strikes are usually designed for retrieving, manipulating or destroying sensitive data or interrupting standard business approaches. Cybersecurity aims to reduce the risk of cyber-attacks and to protect against the illegal manipulation of networks and systems. Implementing robust and reliable measures is challenging due to the ever-evolving innovative attackers.

One of the most noxious security exploits widely used by hackers is Structured Query Language Injection Attacks (SQLiA) according to Open Web Application Security Project (OWASP) Top 10 report 2022 [1]. SQL Injection (SQLi) is an approach that hackers use to acquire illegitimate entryway to a database with the use of malicious input to a database request.

SQL is a computer programming language, used to create, retrieve, update and delete data stored in relational databases. SQL is widely used in all relational database management systems (RDBMS) like MySQL as the standard database language. SQL is advantageous in allowing users to access data, to describe data, define the data and manipulate the data. It also allows user to create and drop databases and tables, to create view, stored procedures and functions, and also setting permissions on tables, views and procedures. SQL allows embedding using modules, libraries and pre-compilers with other languages.

The main reason why SQLi keeps traction is due to improper and insecure development architecture. Making use of legacy software introduces security vulnerabilities that might not be detected with modern software. Using authorized and the latest versions of software is critical to avoiding or minimizing potential security exploits, including SQLiA.

SQLiA mainly occurs through concatenation of a malicious SQL command at the end of a given input query on the user frontend. In recent times, there has been an exponential growth in the amount of dynamic websites, that rely on relational databases to store and manipulate large amounts of data in order to provide a rich user experience.

The most commonly used operation is data retrieval using the SELECT command with the WHERE clause.

SELECT col1, col2 FROM table1 WHERE col1=col2;

Advancement of the previous example can be the use of multiple SQL queries concatenated with a UNION statement, which returns a single table.

SELECT col1 FROM table1 WHERE col1=10 UNION SELECT col2 FROM table2 WHERE col2=col1;

SQLi becomes possible when an attacker manages to exploit an inappropriate input validation at the server-side scripting, resulting in the attacker gaining partial or complete access to the database.

SELECT col1 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 4 different types of SQLiA: Piggy-Backed, Stored Procedures, Union Query, and Alternative Encoding according to Halfond, W. G. et. al. [2] and Wei, K. et. al. [3].

- 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., typical SQL query + ";" + INSERT (or UPDATE, DELETE, DROP) *injected query*
- 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 which would result in in a denial of service (DOS), or execute remote instructions.
  e.g., typical SQL query + "; SHUTDOWN; " *injected query*
- Union Query: An attacker injects a UNION SELECT query to mislead the programme into providing information from a table that is different from the one that was intended as stated by Hwang, D. (2022) [4].
  e.g., typical SQL query + "semicolon" + UNION SELECT *injected query*
- 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.

There are some advanced SQLiA types like Blind SQLi where the attackers devise a strategy to circumvent the lack of error notifications, Fast Flux SQLi where DNS method is used to conceal phishing and malware distribution sites behind a constantly changing network of compromised servers, Compounded SQLi is where the attacker uses a combination of two or more wo or more seizures that target the webpage and it has far-fetching implications.

The detection of these attack types becomes a very important task keeping in mind the potential data breaches that could happen. The detection methods range from checking server logs to monitoring of database errors. The pattern match still remains the most common methods to detect SQL Injection attack. It is a combination of static analysis and run time monitoring.

Another approach to detect SQLiA is by making use of various Machine Learning techniques. Here, in the training phase, we define rules and extract features. The training dataset contains both malicious and non-malicious queries. Efficient feature selection is the most crucial factor in order to obtain higher efficiency.

The advantage of Machine Learning techniques is that it allows us to explore a variety of SQL queries. Also, owing to Machine Learning methodologies, the accuracy value of SQL detection is raised and the false positive rate drops.

II. Literature Survey

SQL Injection 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 [5]. 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 [6]. 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 [7]. 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 [8]. 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 their 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 [9]. 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,Hanmanthu, B. et. al. [9] 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 as well as compare different types of SQLi attacks . [10] 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 [11]. 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, SupportVector Machine (SVM), Boosted Decision Tree, Artificial Neural Network, and Decision Tree [12]. 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 [13]. 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 [14]. 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 [15]. 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 [16]. 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 [17]. 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) [18]. 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 vulner-

abilities by casting the problem as a security capture-the-flag and implementing it as an RL problem [19]. 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 [20]. 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 [21]. 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.

III. Machine Learning

Based on how the model is generated the models can be of the following types:

A. Supervised Learning

Supervised Learning develops a model based on input and output data. This system performs prediction or classification based on statistical insights like f1 score, precision, recall value, accuracy,etc.

B. Unsupervised Learning

Unsupervised learning trains the model on unlabeled data. The model itself learns hidden patterns and insights automatically. It cannot directly perform classification since its goal is to find the underlying structure of data, group the similar data together and then represent the similar data in a compressed format.

C. Reinforcement Learning

Reinforcement learning develops a model using a reward based learning system. In this method the agent performs

auctions using trial and error and learns how to behave in the environment by observing the results. Reinforcement learning is used to deal with problems that are complex in nature and cannot be solved using conventional techniques.

IV. Models

A. Logistic Regression

Logistic regression predicts the probability of a binary event. It uses an s-curve or sigmoid curve to classify the output variables and uses maximum likelihood estimation for accuracy. It performs classification faster than other algorithms.

B. Naive Bayes

Naive Bayes is a classification algorithm that predicts the outcome probability. It calculates probability based on the principle of conditional probability. Naive Bayes generates a fast and highly scalable model that can make quick predictions not requiring much training data. It scales on a linear rate with the number of data points and predictors.

C. Support Vector Machine(SVM)

Support vector machine is deep learning algorithm that performs classification that provides a learning basis for future data processing. Support vector works better when there exists a clear separation margin between the output classes compared to other algorithms. SVM is also relatively memory efficient.

D. Decision Tree Classifier

Decision tree is a data mining technique which is used to identify the strategy best suited to reach the target. Decision trees use a combination of graph and tree data structure in which the path from origin to leaves represent the classification rules.It does not need scaling of data and the missing values does not have a considerable effect.

E. Random Forest

In this method random sampling of data is done and classification is done on each subset using decision tree classifiers. The majority votes gained from different trees are considered and classification is done. Random forest method of classification handles large datasets efficiently and provides better accuracy over a single decision tree.

F. Convolutional Neural Network (CNN)

A convolution network network is a neural network which is designed to process structured arrays of data. It is good at finding patterns and the model consists of more than 20 layers and 3 or 4 convolution layers stacked on top of each other. One of the main advantages of using CNN is it does not require human supervision to detect important distinguishing features.

G. K-Nearest Neighbor(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.

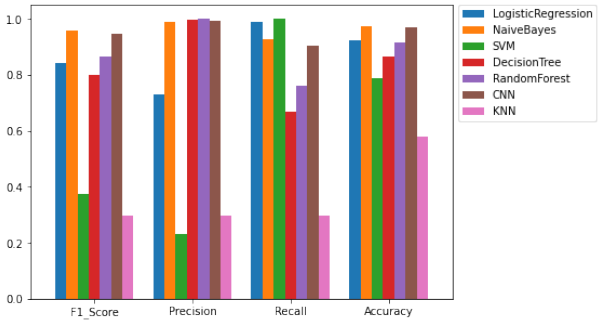


Fig. 1. Comparative result of various Machine Learning models

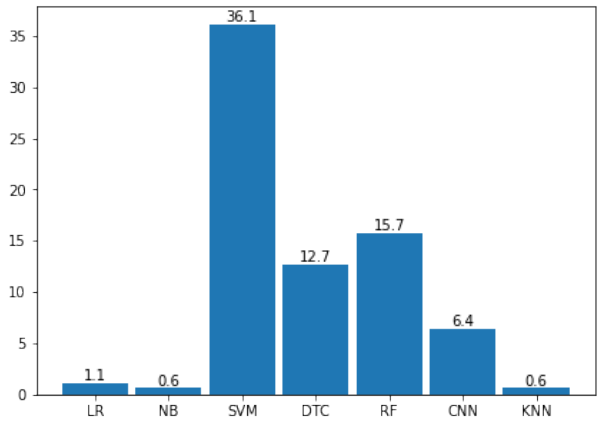


Fig. 2. Compile Time for models

H. Q-Learning

Q-learning is Reinforcement learning method. It is a value-based algorithm that chooses the best action that should be taken at any possible state by estimating the value of each action. The Q- values are progressively updated with the difference that exists between the current estimates of the agent of the actual reward it obtains at run time. Thus the algorithm guarantees obtaining optimal solutions.

V. Experimental Outcomes

We have studied the various types of algorithms used for the detection SQL injection attacks. A comprehensive dataset was used considering different types of SQL attack queries. We have compared multiple models used for detection like Logistic Regression, Naive Bayes, SVM, Decision Tree, Random Fores, CNN and KNN as shown in Fig. 1. Naive Bayes and CNN models are found to show the most consistent accuracy and F1 scores. Fig. 2 shows the time required to compile the models. Naive Bayes and KNN took the least time to compile.

VI. Conclusion

The performance measures of Logistic Regression, Naive Bayes, SVM, Decision Tree, 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 Naive Bayes method. 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 models that can detect SQLi attack.

REFERENCES

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

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

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

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

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

[6] 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).

[7] 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.

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

[9] 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).

[10] 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.

[11] 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

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

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

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

[15] 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

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

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

[18] 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).

[19] 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.

[20] 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.

[21] 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.