Social Sciences & Humanities in Asia (SSHA)

DOI:http://doi.org/10.65098/ssha.01.2026.27.41

ISSN: 3135-3398 (Print)
EISSN: 3135-341X (Online)

RESEARCH ARTICLE

EVALUATION OF MALWARE INJECTIONS IN APPLICATIONS

Adekunle Olayiwola[1,*], Ayeni Oluwatosin Esther[2], Aderoju Olayiwola[3]

[1]Bolmor Polytechnic Ibadan, Oyo State 200284, Nigeria
[2]Ekiti State University, Ado-Ekiti, Ekiti State 360101, Nigeria
[3]Ladoke Akintola University of Technology Ogbomosho, Oyo State 210214, Nigeria
[*]Corresponding Author E-mail: adekunleolayiwola732@gmail.com

ARTICLE DETAILS

Article History:

Received 09 Jun 2025
Accepted 23 Jan 2026
Available online 30 Jan 2026

Online Article Code

ABSTRACT

Background and Purpose: The internet has become integral to daily life, exposing individuals and organizations to increasing malware threats. Malware infections, such as viruses and worms, can lead to unauthorized activities, including data theft, illegal access to databases, and application malfunction. This project aims to develop a Random-4 encryption model integrated with DNA sequencing to detect and mitigate malware injections in applications, enhancing security in data-critical environments.
Methods: The proposed system combines cryptographic techniques with DNA sequencing principles to generate highly randomized encryption keys. By leveraging the randomness of DNA encoding, the system increases the complexity of encryption and improves security. It also incorporates mechanisms for detecting malware injections and authenticating users via email.
Results: The model demonstrates improved data security by utilizing DNA sequencing to generate highly random encryption keys. This approach significantly increases cryptanalysis difficulty, making unauthorized access and malware injections extremely challenging. The system offers stronger resistance to security breaches and enhances overall application integrity.
Conclusion: This work addresses data security challenges by integrating DNA sequencing with Random-4 encryption. The system ensures high randomness and strong protection against malware attacks. Future developments may focus on optimizing the application size and enhancing usability while preserving robust security measures.

KEYWORDS

Malware, Cybersecurity, Encryption, DNA Sequencing, Data Security, Cryptography

1. INTRODUCTION

1.1 Background

The internet has become integral part of life today, which means that nearly everyone faces the threat of malware infections. Malware infection makes local, server or network computers to be infected with one or some of these malicious software (Holt & Bossler, 2013), i.e., viruses, worms, adware, cryptolockers, spyware, toolkits, Trojans, and botnets, which perform unauthorized and destructive activities according to their nature, such as stealing personal or confidential information from users or organizations, tracking and monitoring activities on target computers, installing backdoors and keystroke loggers, crashing target servers, degrading the performance of online systems, consuming system resources such as CPU cycles, memory, and network bandwidth, and encrypting important data to render it inaccessible. The sign of spyware or adware can be a sudden change in running performance. It is important to know that severity of infection varies from system to system, various (rootkits) causing additional destruction and loss than others. If longer malware residues on a system, it much probably downloads more malicious files.

Another emerging phishing attack is Tab Nabbing, which can deceive even tech savvy online users. A survey on web security was conducted by us and a total of about 100 students participated. It is really surprising to note that nearly 80% were unable to identify phishing attack and around 70% could not identify Email spam. Hence there is a need that everyone has basic awareness about web security, since most of the confidential transactions are carried out on the web. In this project, we take up SQL Injection, a critical web security vulnerability. Structured Query Language Injection Attack (SQLIA) is a type of code-injection attack. It is caused mainly due to improper validation of user input. Solutions addressed to prevent SQLIA include existing defensive coding practices alongside encryption algorithms based on randomization. Defensive coding mechanisms are sometimes prone to errors, hence not complete in eradicating the effect of vulnerability. Defensive programming is sometimes very labor intensive, thus not very effective in preventing SQLIA. SQLIA is application-level security vulnerability. According to Abdalla et al., (2022), Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop,

purchase, and maintain applications that can be trusted. Furthermore, according to the OWASP report, malware injection like SQLIA has the highest frequency among other web application attacks which shows the importance of securing web applications from this type of attack by protecting the web applications and their data. Malware attacks have been growing rapidly last 10 years. These attacks targeted all 4-technology device including mobile phones. Due to the personality of the mobile usage and the sensitive data they might contain, safeguards against malwares must be implemented. In this project, we introduced different types of malware injections attacks that is attached to mobile applications.

1.2 Motivation

Malware is a major global problem. It refers to any malicious program designed to disrupt, damage, or gain unauthorized access to computer systems. Studies on malware injection attacks have shown that machine learning–based Structured Query Language (SQL) injection detection systems are effective in preventing the spread of such attacks and protecting users and organizations. Moreover, machine learning techniques can be used to estimate and predict various forms of attacks. These approaches enable the identification and mitigation of SQLIA in online applications, thereby reducing their impact and improving overall system security.

1.3 Objectives

The primary objective of this project is to:

(1) establish relationship between malware injections technique in applications using different machine learning models;

(2) evaluate the effect of different machine learning models above.

1.4 Methodology

Machine learning is a method of computer algorithms that improve automatically through experience. Machine learning algorithms create models based on example data, referred to as training data, in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop traditional algorithms to perform the required tasks (Al-Maliki & Jasim, 2022). More specifically, machine learning processing is based on four key steps, each of which presents its own unique challenges.

(1) Data collection is typically the initial step, an acquisition of raw data, that highly depends on the objective or subjective selection criteria.

(2) Preprocessing, a highly critical step in the data analysis process, shapes data for the neural network to be trained, forming a structured multi-dimensional dataset. Preprocessing may increase or reduce the accuracy of the applied method based on a multitude of factors, discussed later in the survey. For example, duplicates can make the neural model biased.

(3) Choosing, applying, and fine-tuning a machine learning algorithm (e.g., support vector machine, neural network, and decision tree) for a selected task is another challenge, as none of these algorithms is universal. Configuring the selected algorithm becomes a subtask that is primarily based on a trial-and-error approach in order to improve the accuracy of the chosen method.

(4) The last step in the machine learning process is to read the output data, evaluate and present it in a way that can be further used by other systems or visually represented for human understanding. For example, improper selection of a testing dataset can affect the metrics of the output performance.

By analyzing a large amount of SQL injection data, relevant features are extracted, and then the neural network model is trained using a large amount of actual data.

1.5 Significance of Study

The machine learning model helps to classify the queries as SQL injection statement or non-SQL injection statement and has web-code flexibility to detect and have better prevention. It will detect and prevent brute force attack, SQL injection attack, man in the middle attack and malware attacks, thereby increasing knowledge and reduced vulnerabilities risks.

1.6 Organization of Research

Section 1 of this document provides a general introduction to web applications, their vulnerabilities, and how malware injections cause damage and attacks that compromise users and data. It also discusses the advantages of machine learning over other algorithms and outlines the objectives of this project. Section 2 presents a summary of where and how SQLIA and machine learning algorithms are used for classification. It describes the deep learning process and different cross-validation methods. This section also includes an explanation of various SQLIA detection techniques. Section 3 covers the system analysis and design. It involves the system architecture and the mathematical model of the machine learning classifier developed in this study. This section describes the system design and development in detail. Section 4 focuses on system evaluation and results. It provides a description of the dataset used to test the algorithm and presents the results obtained from the test dataset. This section concludes with a comparison of performance using different cross-validation methods. Section 5 concludes the project by presenting the conclusions drawn from experiments conducted using the algorithm on the selected datasets.

2. LITERATURE REVIEW

2.1 Introduction

This chapter reviews various journals, papers, and research works related to the proposed system for evaluating malware injection in web applications. The system models for web applications proposed by researchers to address malware-related problems have been realized through the implementation of malware injection techniques. This literature review examines different malware injection techniques, with a particular focus on SQLIA. Accordingly, the next section provides an overview of malware injection attacks, followed by a section that discusses SQLIA techniques, different types of SQLIA, and the detection methods used to mitigate SQLIA-based malware attacks.

2.2 System Model on Web Application

Web applications are applications that run over a network, such as the Internet or an intranet. They enable websites to become dynamic by establishing connections with databases. The high-level system components of web applications are shown in Figure 1.

In a web application architecture, there are five layers: browsers, networks, web servers, web applications, and databases. First, the client requests a page, either static or dynamic (Ammar & Ansam, 2016). Second, the web browser passes this request through the firewall to the web server. Third, the web server processes the request based on its initial configuration, such as Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS), which handle the request by decoding the webpage. Fourth, the web server forwards the request to the web application server. Finally, the web application sends the request to the database.

In addition, the web application processes commands and verifies secure access to the database through middleware technologies such as Java Database Connectivity (JDBC), SQL for Java (SQLJ), Java Data Objects (JDO) Application Programming Interface (API), and Open Database Connectivity (ODBC). After verifying database access, the web application server sends SQL requests to the database server. Finally, the database server handles the request by allowing data storage, deletion, and updating, depending on the SQL query, and then returns the results to the application server.

Generally, web applications use query statements to generate strings for interacting with databases. These queries are typically generated by web application servers such as Active Server Pages (ASP), JavaServer Pages (JSP), and Hypertext Preprocessor (PHP). A query string

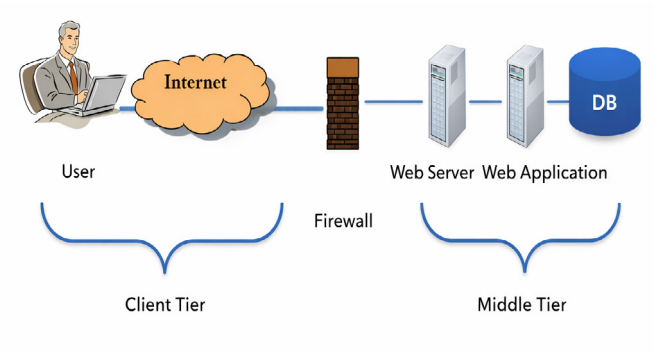

Figure 1 Web Application Architecture

contains both the SQL statement and its parameters, such as a user's name and password. This string is then forwarded to the database server and processed as a single SQL statement. If the received string is compromised or injected with malicious input, it may result in data leakage. Therefore, it is necessary to protect web applications from unauthorized access.

2.3 Malware in Relation to SOL Injections

2.3.1 *Malware in Brief*

Malware is a general term for software inserted into an information system to cause harm to that system or other systems, or to subvert them for purposes other than those intended by their owners. Malware can gain remote access to an information system; record and transmit data from that system to a third party without the user's permission or knowledge; conceal the compromise of the information system; disable security measures; damage the information system; or otherwise affect data and system integrity (Danchev, 2006).

Different types of malware are commonly described as viruses, worms, Trojan horses, backdoors, keystroke loggers, rootkits, or spyware. These terms correspond to the functionality and behavior of the malware (e.g., a virus is self-propagating and whereas a worm is self-replicating). Stephens et al. (2016) classify malware into two categories: family and variant. Family refers to a distinct or original piece of malware, while variant refers to a modified version of the original malicious code, or family, with minor changes.

2.3.2 How does malware work?

Malware is able to compromise information systems due to a combination of factors, including insecure operating system design and related software vulnerabilities. Malware operates by running or installing itself on an information system, either manually or automatically.

Software may contain vulnerabilities, or "holes" in its fabric, caused by faulty coding. Software may also be improperly configured, have certain functionality disabled, be used in a manner inconsistent with recommended practices, or be incorrectly configured in conjunction with other software (Lee, 2019). All of these conditions represent potential vulnerabilities and attack vectors. Once such vulnerabilities are discovered, malware can be developed to exploit them for malicious purposes before the security community has implemented a "fix," commonly known as a patch.

Malware can also compromise information systems due to non-technological factors, such as poor user practices and inadequate security policies and procedures. Many types of malware, such as viruses or Trojans, require some level of user interaction to initiate the infection process, including clicking a malicious link in an email, opening an executable file attached to an email, or visiting a website that hosts malware. Once security has been breached through the initial infection, some forms of malware automatically install additional functionality, such as spyware (e.g., keyloggers), backdoors, rootkits, or other types of malware, collectively referred to as the payload (Emmanuel et al., 2019).

Social engineering, in the form of email messages that are intriguing or appear to originate from legitimate organizations, is often used to persuade users to click malicious links or download malware. A study conducted by Google, which examined several billion URLs and included an in-depth analysis of 4.5 million URLs, found that approximately 700,000 appeared to be malicious and that 450,000 were capable of launching malicious downloads. Another report indicated that only about one in five analyzed websites were malicious by design. These findings led to the conclusion that approximately 80% of web-based malware is hosted on innocent but compromised websites, often without the knowledge of their owners (Bailey et al., 2007).

2.4 Different Malware Injection Approach

Software contains a variety of vulnerabilities that can have a wide range of impacts on an organization. Exploitation of a vulnerable application may result in data breaches, malware infections, Denial-of-Service (DoS) attacks, and other serious threats. Code injection vulnerabilities are particularly dangerous because they can be leveraged by attackers to carry out all of these attacks and more.

Code injection attacks allow malicious code to be executed or injected into a system hosting a vulnerable application. This provides attackers with initial access to an organization's environment and the opportunity to expand their foothold and escalate privileges in order to achieve a range of malicious objectives.

Injection attacks exploit the fact that many applications incorporate user-provided input when constructing commands. For example, in a password-based authentication system, a username may be included in an SQL query to retrieve the corresponding password. Injection attacks use deliberately malformed input to cause user-supplied data to be interpreted as executable code. This is typically achieved by terminating the data portion of the input and altering the intended structure of the command.

For example, consider the following SQL query:

*SELECT * FROM customers WHERE username = '<username>';*

If a user provides the username 'a' OR '1'='1', the resulting query becomes:

*SELECT * FROM customers WHERE username = 'a' OR '1'='1';*

This changes the command executed, and since '1'='1' is always true, the query returns all records in the table. Code injection attacks come in a variety of forms. Some common types of code injection attacks include:

(1) SQL Injection: As demonstrated above, malicious input can alter the interpretation of an SQL command. This can be exploited to access unauthorized records, modify the database, or perform other malicious actions.

(2) Cross-Site Scripting (XSS): XSS attacks exploit the fact that executable scripts can be embedded within HTML content. Websites that incorporate user-provided data into webpages without proper

validation can be manipulated to execute malicious scripts within a user's browser.

(3) Command Injection: Some applications are designed to incorporate user-provided data into commands executed by the system shell. Attacker-supplied input can modify the intended command or terminate it and execute arbitrary shell commands chosen by the attacker.

(4) File Injection: Some applications accept and process user-provided or user-selected files. Exploitation of vulnerable code may allow an attacker to execute malicious code embedded within such files.

(5) Deserialization: Some applications serialize multiple variables into a single string to simplify data transmission. Malicious input may exploit vulnerabilities in the deserialization logic or cause user-provided data to be interpreted as part of other fields, potentially influencing code execution.

(6) Software Supply Chain Attacks: Software supply chain attacks, such as NotPetya, demonstrate the severe damage that malicious code injection can cause. In such attacks, adversaries compromise a trusted network or update mechanism and inject malware into legitimate software, thereby enabling unauthorized access to systems and resources.

2.5 SQL Injection and Detection Methods

2.5.1 SQL Injection

SQL injection is an attack in which an attacker exploits vulnerabilities in a database application to manipulate SQL statements, thereby bypassing database security mechanisms and gaining unauthorized access to the database. Through SQL injection, attackers can perform a series of malicious operations, such as inserting, deleting, modifying, or retrieving database records (McWhirter et al., 2018)

Specifically, an attacker exploits these vulnerabilities by supplying carefully crafted input data, causing the SQL interpreter to be unable to distinguish between legitimate query commands and malicious input. Because the SQL language is expressive and supports diverse encoding methods, databases are highly susceptible to attacks during the dynamic construction of SQL statements (Joanna et al., 2022).

In many cases, developers do not adequately validate or filter user input. When an attacker appends maliciously constructed URL parameters or form submission data to predefined SQL queries, the intended SQL structure can be altered. As a result, the executed SQL statements produce outcomes that differ from those expected by the developer, thereby forming a SQLIA (Wei et al., 2022).

2.5.2 SQLIA

SQLIAs are among the most common hacking techniques in use today. During such attacks, an attacker can compose, read, modify, or delete data stored in a database. This type of attack allows the attacker to transform an originally legitimate SQL query into malicious code, enabling the extraction of sensitive information or the corruption of data within the database. In an SQLIA, the attacker injects malicious code into input fields intended for legitimate user data in a web application, thereby bypassing authorization mechanisms and gaining unrestricted access to database resources (Choudhary et al., 2016).

An attacker can extract sensitive information, modify data, or even destroy all data stored in a back-end database. SQL injection exploits vulnerabilities at the database layer and is a relatively straightforward attack technique in which attackers insert malicious SQL code into legitimate queries. This enables them to retrieve sensitive data or compromise the integrity of the entire database.

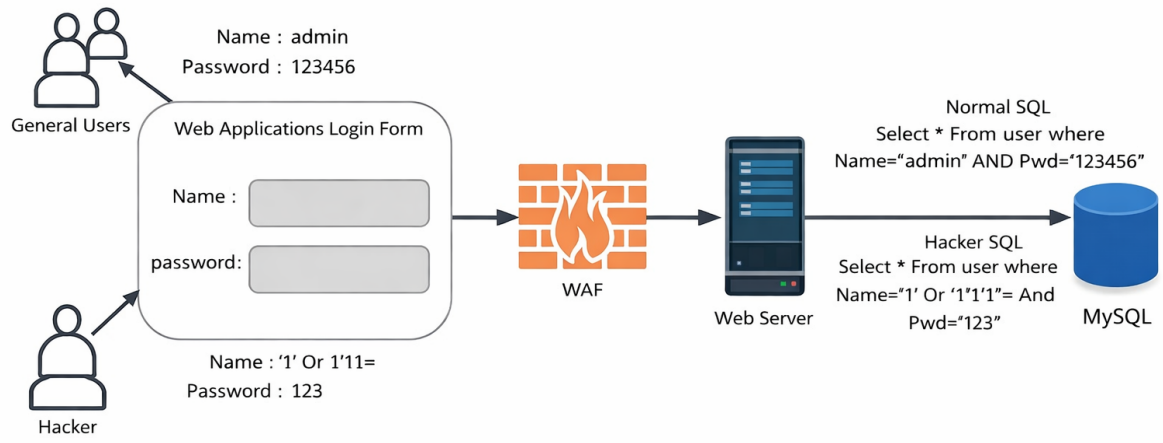

Figure 2 Procedure of SQLIA

2.5.3 Procedure of SQLIA

The SQL injection process works by prematurely terminating the intended query context and appending a new malicious statement. Because the injected statement becomes part of the original query before execution is completed, the attacker is able to alter the intended logic of the SQL command. Generally, the SQL injection process consists of three main stages, as shown in Figure 2 (Choudhary et al., 2016). First, an attacker sends malicious HTTP input to the web application. The application then constructs SQL queries using this input and forwards them to the back-end database to retrieve, manipulate, and present the requested data.

2.5.4 Classification of SQLIA

In SQLIA, an attacker attempts to alter the logic of an SQL query by injecting additional malicious SQL code. As a result, SQLIA can be classified into several categories based on the attack techniques employed (Choudhary et al., 2016). These categories include the following:

(1) Tautologies: This type of attack is commonly used to bypass user authentication mechanisms and retrieve unauthorized data by inserting tautological conditions into SQL statements. A tautology is a condition that always evaluates to true, thereby forcing the query to return all records.

*Example: select * from Student where name= ' ' or 1=1 -- and Roll = ' ';*

(2) Logically Incorrect Queries: In this type of attack, the attacker intentionally submits malformed or logically incorrect SQL queries to gather information about the database schema and back-end architecture. When an invalid query is executed, the database may return error messages containing details such as table names, column names, or data types, which can be exploited for further attacks.

Example: Collective objects activated on Varchar and reasonless datatypes;

(3) Union Query: In this type of attack, the attacker establishes supplementary statement into the primary SQL statement. This attack can be completed by set up either a UNION query or a comment of the form into accessible framework.

*Example: select balance from Account where candid= '123' UNION select * from Account;*

(4) Stored procedure: Stored procedures are predefined sets of SQL statements stored in the database and executed as a single unit. Although they are often considered more secure, stored procedures can still be vulnerable to SQL injection if they improperly handle user input. In a Stored Procedure SQLIA, the attacker injects malicious SQL code, such as additional commands or control statements, into input parameters, which may alter the execution flow of the procedure.

Example: select marks from Result where name = 'Rahul' and id= '101'; SHUTDOWN;

(5) Piggy-Backed Queries: In this type of attack, the attacker attempts to append additional malicious queries to an originally legitimate SQL query. This query-based attack is particularly dangerous because it allows the attacker to inject and execute multiple SQL statements within a single database request. The malicious queries are embedded within valid user input and are executed together with the original query by the database.

*Example: select * from Employee where empid= '1111' and pass= '1501'; drop table Employee;*

(6) Inference: Inference-based SQLIA exploit the behavior and responses of the database system to infer sensitive information, even when direct query results are not returned. In this type of attack, the attacker observes the application's responses to carefully crafted inputs to deduce the structure and content of the database.

Example: select emp_name, emp_address, gender from employee where '1'= '0';

(7) Blind Injection: In a Blind SQL Injection attack, the attacker does not receive direct feedback from the database in the form of error messages or query results. Instead, the attacker infers information about the database by submitting a series of specially crafted SQL queries that return Boolean (true/false) responses. By analyzing the application's behavior in response to these queries, the attacker can gradually extract sensitive information, even when error messages are suppressed by the application design.

*Example: select * from User where id='45' and '1'='0';*

(8) Time-based Attack: In a Time-based SQLIA, the attacker extracts information from the database by analyzing response time delays. This type of attack relies on conditional SQL statements that deliberately introduce time delays (e.g., using IF-THEN logic or database sleep functions) when a specific condition is true. By measuring these delays in the database responses, the attacker can infer sensitive information about the underlying database structure and data.

*Example: select * from Table where id='101' or pass= '1'='0';*

2.5.5 Impact of SQLIA

SQLIAs have a significant impact on information security due to breaches of confidentiality involving data stored in databases. The loss of confidentiality, along with the associated financial costs related to system recovery, service downtime, regulatory penalties, and reputational damage, represents the primary immediate consequences of a successful SQLIA. Table 1 summarizes the impacts of successful SQLIAs (Ammar & Ansam, 2016).

2.6 SQLIA Methods

2.6.1 Traditional Methods

Traditional SQLIA detection methods mainly rely on filtering special characters to prevent SQLIA. In SQLIA defense, whitelist validation and blacklist validation are identified as two common input validation techniques used to prevent SQLIA (Clarke, 2009). Although penetration testing can compensate for certain limitations of blacklist- and whitelist-based filtering mechanisms, it cannot fundamentally resolve their inherent weaknesses (Tian et al., 2012).

Halfond and Orso (2005) developed a tool called AMNESIA, which is based on traditional blacklisting techniques and implements a combined static and dynamic analysis approach. In the static phase, AMNESIA automatically constructs a model of legitimate SQL queries that can be generated by the application. In the dynamic phase, it examines runtime-generated queries and checks them against the statically constructed model.

Xiao et al. (2017) proposed a SQLIA detection method based on URL–SQL mapping to analyze user behavior and execution responses. This method incurs relatively low additional overhead for web applications; however, system execution involves uncertainty, and the extraction of invariants (i.e., the normal operational state of the web application) is not comprehensive. As a result, the method may generate a high number of false positives and false negatives.

Overall, while these traditional methods were effective when SQLIA primarily relied on simple string-matching techniques, modern attackers increasingly employ sophisticated automated injection tools. Consequently, traditional SQLIA detection approaches often fail to cope effectively with contemporary attack patterns (Wei et al., 2022).

2.6.2 Machine Learning

In formal terms, machine learning can be defined as a subfield of Artificial Intelligence (AI) in which systems acquire intelligence without explicit programming. Human beings gain knowledge for various tasks through learning; inspired by this observation, research in AI gradually shifted over one to two decades toward developing algorithms that improve their performance based on data. The primary objective of this subfield has been to design algorithms capable of acquiring relevant knowledge for a given task or problem domain from data. It is important to note that this knowledge acquisition typically relies on labeled data and an appropriate representation of such data, which is defined by human experts.

The application of machine learning to SQLIAs has become increasingly common. Joshi and Geetha (2014) designed a classifier that combines a Naive Bayes machine learning algorithm with a role-based access control mechanism to detect SQLIAs, and evaluated the model using three types of SQLIA attacks: annotation, union, and restatement. Kamtuo and Soomlek (2016) proposed a decision tree–based SQLIA prevention framework and trained the machine learning model using 1,100 vulnerability datasets.

Wu and Chen (2012) proposed a method called k-centres (KC) for SQLIA detection. This approach is based on the k-means clustering algorithm applied to hybrid data and adapts the number and location of different attack types within KC clusters to improve detection performance.

However, machine learning-based SQLIA detection methods also have inherent limitations. McWhirter et al. (2018) noted issues such as the inability to detect attacks that employ injection evasion techniques, susceptibility to overfitting, and the requirement for manual feature filtering. Moreover, many existing approaches rely primarily on raw SQL query string analysis and therefore fail to fully exploit recent advances in machine learning, as well as the contextual and syntactic information contained in SQL statements (Zhuo et al., 2021).

2.6.3 Deep Neural Networks

Deep neural networks, also referred to as deep learning, are a recent research direction within the field of machine learning. They were introduced to bring machine learning closer to its original goal of AI, as illustrated in Figure 3 (Chen, 2014).

Table 1 Impacts of Successful SQLIA

Impacts	Explanation
Authentication Bypass	This attack allows SQLI attacker to get access to a database layer, possibly with admin privileges, without providing a correct username or password.
Information Disclosure	This attack allows SQLI attacker to gain for sensitive information that is stored in a database such as credit card information.
Compromised Data Integrity	This attack allows SQLI attacker to modify the contents of web page. The consequence from this is defacing a web page.
Compromised Availability of Data	This attack allows SQLI attacker to remove information in order to cause damage to information that is stored in a database.
Remote Command Execution	This attack allows SQLI attacker to perform command execution through a database, which let the attacker control for the operating system.

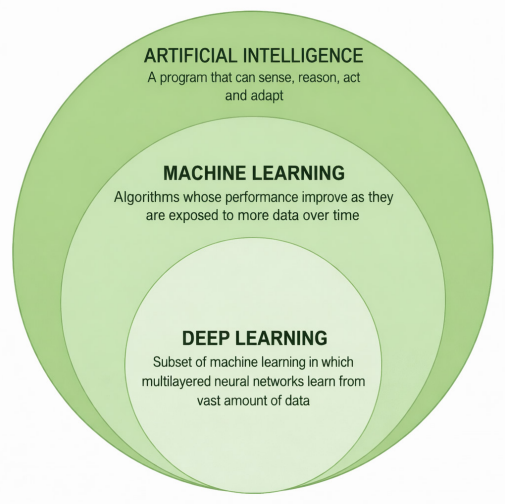

Figure 3 Deep Learning Family

Recently, machine learning has become increasingly widespread in research and has been incorporated into a wide range of applications, including text mining, spam detection, video recommendation, image classification, and multimedia concept retrieval (Alzuaidi et al., 2021). Among the various machine learning algorithms, deep learning is one of the most commonly employed approaches in these applications, as illustrated in Figure 4.

Deep learning is derived from conventional neural networks but significantly outperforms earlier neural network models. Moreover, deep learning simultaneously leverages hierarchical transformations and graph-based structures to construct multi-layer learning models. Recent advances in deep learning techniques have achieved outstanding performance across numerous application domains, including audio and speech processing, visual data analysis, and natural language processing (Alzubaidi et al., 2021).

Deep learning focuses on learning intrinsic patterns and hierarchical-representations from sample data, and the knowledge acquired through this process can greatly facilitate the interpretation of complex data such as text, images, and audio signals (Chen, 2014). The ultimate goal of deep learning is to enable machines to acquire analytical learning capabilities similar to those of humans, allowing them to recognize and understand data in various forms, including text, images, and sounds.

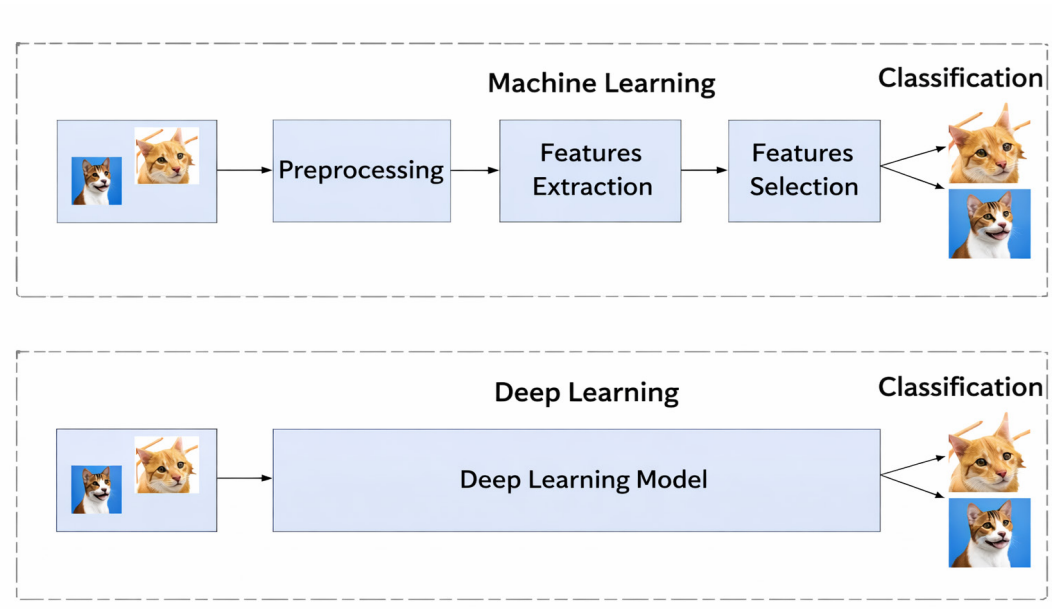

Figure 4 The Difference Between Deep Learning and Traditional Machine Learning

Deep learning is a complex machine learning algorithm that has achieved results in speech and image recognition that far exceed previous related techniques. Deep learning has also been successively applied to web security detection in recent years. Sirinam et al. (2018) used fingerprinting attacks on a Convolutional Neural Network (CNN) defence website and showed that the accuracy of CNN for fingerprinting attack detection on this website was above 98%. Yuan et al. (2017) provided a subspace spectrum integration clustering method that supports deep learning for web attack detection. Selvaganapathy et al. (2018) used deep confidence networks to extract URL features and used deep neural networks to classify normal and malicious URLs.

2.7 SQL Injection Characterization

SQL injection detection can be formulated as a binary classification problem. In this task, data samples are categorized into two primary classes: SQL injection statements and non-SQL injection statements. The non-SQL injection statements further include ordinary SQL statements and ordinary text statements. In the dataset, SQL injection statements are labeled as 1, while non-SQL injection statements are labeled as 0.

The primary objective of SQL injection detection is to distinguish malicious SQL injection statements from non-malicious inputs. Ordinary SQL statements and ordinary text statements are therefore grouped into a single non-aggressive category. This formulation enables the classification model to effectively differentiate aggressive SQL injection statements from non-aggressive ones.

Based on the scope of this study, the dataset used in this project contains a total of 30,919 samples, which are divided into the two categories described above.

2.7.1 SQL Injection Statements

During the current study, SQL injection statements were observed to frequently contain a variety of special characters, including "*, ;, _, -, (), =, {}, @, ., [], +, ?, %, !, :, \, and /". These characters are commonly used in SQL injection payloads to manipulate query structure or terminate legitimate statements.

In addition, SQL injection statements often include common SQL keywords, operators, and functions. Examples include Where, Table, Like, Select, Update, And, Or, Set, In, Having, Values, Into, Alter, As, Create, Revoke, Deny, Convert, Exec, Concat, Char, Ascii, Any, Asc, Desc, Check, Group By, Order By, Delete From, Insert Into, Drop Table, Union, and Join.

Based on the dataset used in this project, SQL injection statements are further classified into the following types:

(1) Repetition attack: the main purpose is to detect whether there is an injection point by the truth or falsity of the expression. The SQL language contains keywords such as where and group by, which are generally followed by query conditions. The attacker inserts the true-true formula into the position of the conditional statement so that the query condition is always identified as true, which can bypass the page authentication and obtain sensitive data.

(2) Illegal comment attack: SQL language contains various forms of comments, such as "--, #, /, *, and /", whose subsequent statements will not be executed. The attacker often inserts these comment characters into the conditional SQL statement so that the legitimate SQL statement after the comment character cannot be executed to achieve the injection purpose.

(3) Union query attack: in SQL language, keywords such as join, left join, and union are used for the union implementation of one or more statements. An attacker can use the union keyword to add malicious SQL code after the legitimate code to obtain additional sensitive information by controlling the second statement to perform illegal operations.

(4) Explicit error injection: the attacker makes the application display default error pages by executing illegal or logically incorrect SQL queries. These pages often reveal to the attacker information such as injectable parameters and the type of database.

(5) Boolean blind injection: the main purpose is to submit queries with a different logic and observe whether the page returned by the web application is normal to determine whether there is an injection. Boolean blinding is often used to obtain information such as the name of the database.

(6) Time blind injection: it mainly refers to inserting a time delay function in the URL or user input to determine whether the injection is successful by observing whether there is a delay in the web application's response.

(7) Multi-statement attack: this attack generally uses query separators to add additional queries to the original query to extract, add, and modify data or execute remote commands. The DBMS receives multiple SQL queries, the first is the normal execution of the query, and the subsequent queries are executed to meet the attack. SQL injection statements using the above keywords and injection types can bypass the security defences of the database and enter the database with a false identity. After the database is invaded, it does not detect such enemies by itself and only allows such enemies to wreak havoc in the database, such as adding, deleting, modifying, and viewing tables.

2.7.2 Non-SQL Injection Statements

(1) Plain text statements: ordinary text statement is composed of ordinary letters, numbers, and characters. The statement is not offensive in any way and will be recognized directly in the detection as a nonformal statement.

(2) General SQL statements: general SQL statements are the SQL statements that users encounter every day and are used in the daily maintenance of the database, for example, creating, querying, and modifying tables. These types of SQL statements usually contain keywords such as rename, drop, delete, insert, create, exec, update, union, set, alter, database, and, or, information_schema, load_file, select, shutdown, cmd shell, hex, ascii, etc. It will also contain some dangerous keywords: "--, #, /* , ', ", ||, \\, =".

2.7.3 Data Characterization

Analysis of the data shows that words such as "select", "*", "from" appear very frequently in SQL injection statements and non-SQL injection statements, and these very frequent words do not serve as features for classification. Therefore, in this project, when using the TF-IDF algorithm to process the data, such words can be filtered out sufficiently to achieve dimensionality reduction.

2.8 Past Related Works and Reviews

In this section, relevant studies are reviewed with a focus on their motivations, objectives, methodologies, contributions, and limitations.

2.8.1 Malware and Attack Technologies Knowledge Area

Malware, short for malicious software, refers to any program designed to perform malicious activities. In this work, the terms malware and malicious code are used interchangeably (Lee, 2019). Malware exists in a wide variety of forms and can be classified into different categories, including viruses, Trojans, worms, spyware, botnet malware, and ransomware. Malware is responsible for a large proportion of cyberattacks on the Internet, ranging from nation-state cyber warfare to cybercrime, fraud, and online scams.

Trojan malware can introduce backdoors into government or enterprise networks, enabling nation-state attackers to gain unauthorized access and steal classified information. Ransomware encrypts data on a victim's computer, rendering it inaccessible until a ransom is paid for decryption. Botnet malware is widely used to conduct Distributed Denial-of-Service attacks, as well as spam distribution and phishing campaigns.

Given the growing prevalence and impact of such threats, it is necessary to study the techniques underlying malware development and deployment in order to better understand cyberattacks and to design effective countermeasures. As political and financial stakes continue to rise, both malware technologies and cyber defense mechanisms have evolved in sophistication and robustness. This work is organized as follows: it first provides a taxonomy of malware and discusses their typical malicious

activities, as well as their ecosystems and supporting infrastructures. It then describes tools and techniques for malware behavior analysis, along with network-based and host-based detection methods for identifying malware activities. Finally, it examines response processes and techniques, including forensic analysis and attack attribution.

2.8.2 Deep Neural Network-Based SQL Injection Detection Method

Among network security threats, SQL injection is a common and challenging form of attack that can cause severe and potentially incalculable damage to databases. Consequently, the detection of SQL injection statements has become a major research focus in recent years. Based on the structural and semantic characteristics of SQL statements, Wei et al. (2022) proposed a SQL injection detection model and algorithm based on a deep neural network.

The core idea of the proposed approach is to transform SQL statement data into word vector representations using a tokenization method, construct sparse matrices from these vectors, and input them into the model for training. A deep neural network (DNN) architecture with multiple hidden layers and Rectified Linear Unit activation functions is employed. In addition, the traditional loss function is optimized, and a Dropout mechanism is introduced to enhance the model's generalization capability. Experimental results show that the final model achieves an accuracy exceeding 96%.

Comparative experiments with traditional machine learning algorithms and Long Short-Term Memory (LSTM) models demonstrate that the proposed approach effectively addresses common issues such as overfitting and the reliance on manual feature extraction in conventional machine learning methods. As a result, the model significantly improves the accuracy of SQL injection detection.

2.8.3 SQLIA Detection and Prevention Techniques Using Machine Learning

Web application attacks are continuously increasing in both frequency and severity. The vast amount of data available on the Internet has motivated attackers to launch increasingly sophisticated attacks. In this context, extensive research has been conducted on web application security. Among the various threats targeting web applications, SQL injection is considered one of the most dangerous attacks, posing a serious risk to web-based systems.

Numerous studies have focused on mitigating SQL injection attacks, either by preventing them at an early stage or by detecting them during execution. Jemal et al. (2020) presented a comprehensive overview of SQL injection attacks and provided a classification of recently proposed detection and prevention approaches. Their work categorizes SQL injection attacks according to attack sources, objectives, and types. Furthermore, it reviews and classifies the most important and recent mitigation techniques, with particular emphasis on approaches based on ontology and machine learning.

2.8.4 SQLIA Predictive Analytics Using Supervised Machine Learning Techniques

SQLIA is one of the most prevalent cyberattacks targeting vulnerabilities in web-based applications. Such attacks exploit injection techniques to gain unauthorized access to restricted data, bypass authentication mechanisms, and execute unauthorized data manipulation language operations. Numerous solutions and approaches have been proposed for the identification and prevention of SQLIA, including cryptography, Extensible Markup Language (XML), pattern matching, parsing techniques, and Machine Learning–based methods.

Among these approaches, Machine Learning has been shown to be particularly effective for SQLIA mitigation when implemented within a defensive coding framework. Machine Learning approaches require large volumes of data for efficient model training and are capable of learning multiple attack patterns, including complex blind SQL injection attacks that are difficult to detect using traditional techniques. Akinsola et al. (2020) conducted an experimental analysis using the Waikato Environment for Knowledge Analysis to evaluate several supervised classification algorithms, including Logistic Regression (LR), Stochastic Gradient Descent (SD), Sequential Minimal Optimization (SMO), Bayes Network (BN), Instance Based Learner (IBK), Multilayer Perceptron (MLP), Naive Bayes (NB), and J48. Hold-Out (70%) and 10-fold Cross-Validation evaluation techniques were employed to assess algorithm performance.

The Cross-Validation results showed that SMO, IBK, and J48 achieved accuracies of 99.982%, 99.995%, and 99.999%, respectively, while the Hold-Out evaluation reported accuracies of 99.986%, 99.989%, and 100% for the same algorithms. In terms of model construction time, Cross-Validation results indicated that SMO, IBK, and J48 required 10.15s, 0.06s, and 14.12 s, respectively, whereas the Hold-Out evaluation showed corresponding times of 9.71s, 0.16s, and 14.28s.

Based on these findings, IBK demonstrated the shortest model construction time under Cross-Validation, along with superior performance in accuracy, sensitivity, and specificity, and was therefore selected as the preferred classifier for SQLIA detection and prevention. The study further highlights that, beyond accuracy, additional performance evaluation metrics are essential for optimal algorithm selection in predictive analytics.

2.8.5 Review of SQLIAs: Detection, to Enhance the Security of the Website from Client-Side Attacks

The importance of cybersecurity in protecting data and information has become increasingly significant in the modern technological era. As the number of cyberattacks continues to rise on a daily basis, security systems have undergone continuous development, driven by the growing need to predict, prevent, and mitigate such attacks. Among the top ten security threats identified by the Open Web Application Security Project (OWASP), injection attacks are ranked as one of the most critical risks.

Among these, SQL injection is the most common and one of the most dangerous vulnerabilities, due to its diverse attack types and the rapid, severe consequences it can cause. SQLIAs may result in financial losses, data leakage, extensive database damage, and even complete paralysis of web applications. Despite significant research efforts, detecting SQL injection attacks remains a challenging task. Consequently, the effective defense against SQL injection has become a major research focus and a frontier topic in web application security in recent years.

Machine learning techniques have demonstrated considerable success in addressing these threats by enabling effective prevention and detection of attacks such as cross-site scripting and SQL injection in web applications. Machine learning is widely used to analyze and identify security vulnerabilities by learning patterns from data. It employs both traditional machine learning algorithms and deep learning models to evaluate classification performance based on input validation features, thereby enhancing the accuracy and robustness of web application security systems (Manar & Mahdi, 2022).

2.8.6 SQL Injection Detection Using Machine Learning Techniques and Multiple Data Sources

SQL injection continues to be one of the most damaging security exploits, leading to significant personal information exposure and substantial financial losses. Injection attacks are ranked as the number one vulnerability in the most recent OWASP Top 10 report, and the frequency of such attacks continues to rise. Traditional defense strategies typically rely on static, signature-based Intrusion Detection System rules, which are effective primarily against previously known attacks but perform poorly when confronted with unknown or zero-day attacks.

As a result, much recent research has focused on the application of machine learning techniques, which are capable of identifying previously unseen attack patterns. However, depending on the algorithm employed, these approaches may incur significant performance overhead. Furthermore, most existing intrusion detection strategies collect traffic data either from network devices or directly from the web application host, while alternative approaches rely on database server logs. In this project, traffic is collected from two distinct points: the web application host and a Datiphy appliance node positioned between the web application

host and the associated MySQL database server. An analysis of these two datasets, along with a third dataset generated by correlating the two sources, demonstrates that the accuracy achieved using the correlated dataset with rule-based and decision tree algorithms is comparable to that obtained using neural network algorithms, while achieving significantly improved performance efficiency (Ross, 2018).

2.8.7 Performance Evaluation of Machine Learning Algorithms for Detection and Prevention of Malware Attacks

Malware refers to any type of software intentionally designed to disrupt, damage, or gain unauthorized access to computer systems and networks. Common examples of malware include bots, ransomware, adware, keyloggers, viruses, Trojan horses, and worms. The exponential growth of malware poses a significant threat to the security of confidential information. One of the major limitations of many existing malware classification algorithms is their relatively low performance in accurately detecting and preventing malware infections. Consequently, there is an urgent need to evaluate the effectiveness of existing machine learning classification algorithms used for malware detection.

This study aims to support the development of more robust and efficient detection models capable of overcoming the limitations of current approaches. To achieve this, the performance of several classification algorithms was evaluated, including J48, Logistic Model Trees, Naïve Bayes, Random Forest, Multilayer Perceptron Classifier, Random Tree, Reduced Error Pruning Tree, Bagging, AdaBoost, KStar, Simple Logistic, Instance-Based k-Nearest Neighbor, Locally Weighted Learning, Support Vector Machine, and Radial Basis Function Network. The algorithms were assessed using multiple evaluation metrics, including Accuracy, Precision, Recall, Kappa Statistic, F-Measure, Matthews Correlation Coefficient, Receiver Operating Characteristic Area, and Root Mean Squared Error, employing the WEKA machine learning and data mining toolkit. Experimental results demonstrate that the Random Forest algorithm achieved the highest detection accuracy of 99.2%, indicating its strong effectiveness in malware detection tasks (Emmanuel et al., 2019).

2.8.8 SQL Injection Prevention in Web Application: A Review

A web application is a software system that provides users with an interface through a web browser across different operating systems. Despite the growing popularity of web applications, security threats targeting these systems have become increasingly diverse and severe. Poorly designed web applications are particularly vulnerable to malware attacks, among which SQL injection attacks are the most common. Although SQL injection has been a known vulnerability for over two decades, it remains a critical security concern. The referenced study summarizes fourteen different types of SQLIAs and examines their consequences for web applications. It provides a comprehensive overview of SQLIAs, including their causes, impacts, and attack variants. To effectively defend web applications against SQLIAs, preventive techniques are a crucial component of application security. Preventing SQL injection requires the identification and implementation of appropriate security measures to mitigate potential attack impacts.

In general, SQLIA prevention mechanisms should focus on validating and sanitizing user inputs, encrypting sensitive data, and keeping database systems updated with the latest security patches to prevent attackers from exploiting vulnerabilities in outdated software versions. Based on previous studies, the paper investigates and categorizes existing security mechanisms used to prevent SQLIAs in web applications. The primary objective of the study is to examine various SQL injection prevention methods and to analyze the most effective preventive mechanisms against SQLIAs. For future research, the study proposes evaluating these prevention techniques in practical environments, such as their implementation in other web application domains, including e-commerce platforms, educational systems, and corporate business websites (Joanna et al., 2022).

2.8.9 Web Application Protection against SQL Injection Attack

Web applications have become an essential and integral part of internet usage today as they provide the convenience of business and personal transactions anywhere anytime. However, SQLIAs pose a serious threat to web applications hence; the primary purpose of the research was to present a new model to protect web applications against SQLIAs with least modification of the Web architecture. The proposed model was developed based on negative tainting and SQL syntax-aware methods, and wase valuated through SQL penetration testing in web application and database server. We were able to successfully distinguish between legitimate SQL queries and malicious ones that had adopted various evasion methods such as encoding, comments and white space evasion methods as well as logical expressions and string techniques that were not captured by commercially available detection engines. Future work entails testing our model for various other obfuscations with a larger dataset and with a more comprehensive vulnerabilities look up table (Ammar & Ansam, 2016).

2.8.10 Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions

In recent years, deep learning has been regarded as the gold standard within the machine learning community. It has gradually become the most widely adopted computational paradigm in the field of machine learning, achieving outstanding performance on a variety of complex cognitive tasks and, in some cases, matching or even surpassing human-level performance. One of the major advantages of deep learning lies in its ability to effectively learn from massive amounts of data. The field of deep learning has grown rapidly over the past few years and has been extensively applied to successfully address a wide range of traditional and emerging applications. More importantly, deep learning has consistently outperformed well-established machine learning techniques across numerous domains, including cybersecurity, natural language processing, bioinformatics, robotics and control, and medical information processing, among others.

Although several studies have contributed reviews of the state of the art in deep learning, most of them focus on a single aspect of the field, resulting in a fragmented understanding of deep learning as a whole. To address this limitation, this work adopts a more holistic perspective, providing a suitable starting point for developing a comprehensive understanding of deep learning. In particular, this paper highlights the importance of deep learning and presents the main types of network architectures. It then focuses on convolutional neural networks, which represent the most widely used DNNs, and describes the evolution of CNN architectures along with their key characteristics, beginning with AlexNet and concluding with the High-Resolution Network (HRNet). Furthermore, the paper discusses the major challenges associated with deep learning and proposes potential solutions to help researchers identify existing research gaps. This is followed by a review of major deep learning applications.

Finally, the paper summarizes the computational tools commonly used in deep learning, including field-programmable gate arrays (FPGA), graphics processing units (GPU), and central processing units (CPU), and discusses their influence on deep learning performance. The review concludes with an evolution matrix, benchmark datasets, and a summary and conclusion (Alzubaidi et al., 2021).

2.8.11 Conceptual Understanding of CNN model

Deep learning has become an area of interest to the researchers in the past few years. CNN is a deep learning approach that is widely used for solving complex problems. It overcomes the limitations of traditional machine learning approaches. The motivation of this study is to provide the knowledge and understanding about various aspects of CNN. This study provides the conceptual understanding of CNN along with its three most common architectures, and learning algorithms. This study will help researchers to have a broad comprehension of CNN and motivate them to venture in this field. This study will be a resource and quick reference for those who are interested in this field (Sakshi et al., 2018).

3. SYSTEM ANALYSIS AND DESIGN

3.1 Introduction

In the system analysis and design phase, this paper interprets all the facts gathered from the research, identified problem, decomposed

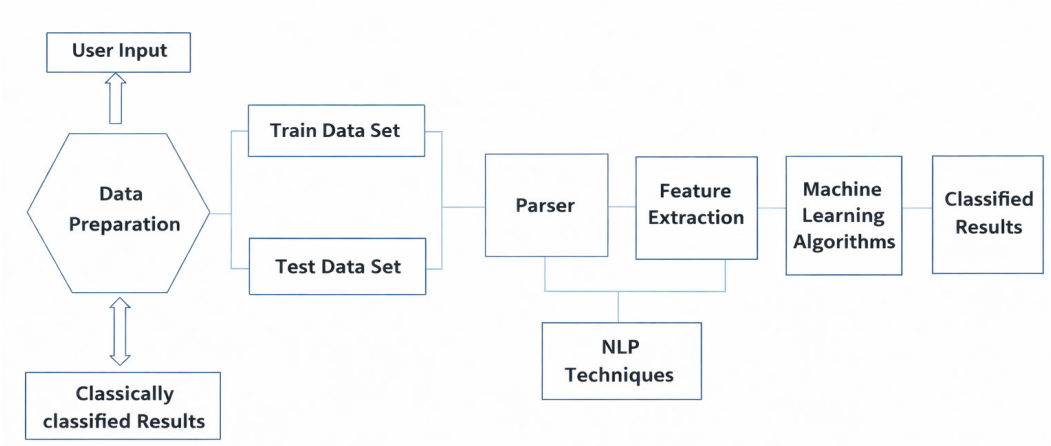

Figure 5 Block Diagram for Proposed Model

the system into its components, and presented the proposed system overview, system architecture, the data flow diagram of the system, the various software tools and the methodology applied for the research.

The system design objective is to evaluate SQL injections attack in applications using machine learning algorithms and showcase their performance relationship with deep learning algorithms. Web applications use query statements to generate strings to interact with the database and the server for checking as a single SQL statement. The uniqueness of this project is the fact that the received string compromised or injected it will cause data leakage. Therefore, it is necessary to protect web applications from illegal accesses.

3.2 Machine Learning Algorithms

Machine learning is a sub-field of artificial intelligence that gives machines the skills to learn from data and predicts the optimum model. ML techniques can be grouped into three main categories Supervised Learning, Unsupervised Learning and Reinforcement Learning. The Supervised Learning method is based on a set of input data X which deals to the result Y, while in the unsupervised learning, the system only has input data X without being told the expected output result Y.

Mathematically, classification is the task of approximating a mapping function (f) from input variables (X) to output variables (Y). It is basically belonging to the supervised machine learning in which targets are also provided along with the input data set. An example of classification problem can be the spam detection in emails. There can be only two categories of output, "spam" and "no spam"; hence this is a binary type classification. To implement this classification, we first need to train the classifier. For this example, "spam" and "no spam" emails would be used as the training data. After successfully train the classifier, it can be used to detect an unknown email.

3.3 Mathematical Model

3.3.1 Naïve Bayes Algorithms

Naïve Bayes algorithms is a classification technique based on applying Bayes' theorem with a strong assumption that all the predictors are independent to each other. In simple words, the assumption is that the presence of a feature in a class is independent to the presence of any other feature in the same class. For example, a phone may be considered as smart if it is having touch screen, internet facility, good camera etc. Though all these features are dependent on each other, they contribute independently to the probability of that the phone is a smart phone.

In Bayesian classification, the main interest is to find the posterior probabilities i.e., the probability of a label given some observed features, $P\ (L \mid features)$. With the help of Bayes theorem, we can express this in quantitative form as follows:

$$P(L \mid \text{features}) = \frac{P(L)(\text{features} \mid L)}{P(\text{features})} \qquad (1)$$

where $P\ (L \mid features)$ is the posterior probability of class, $P(L)$ is the prior probability of class. $P\ (features \mid L)$ is the likelihood which is the probability of predictor given class. $P\ (features)$ is the prior probability of predictor.

3.3.2 Building model using Naïve Bayes in Python

Python library, Scikit learn is the most useful library that helps us to build a Naïve Bayes model in Python. Gaussian Naïve Bayes: it is the simplest Naïve Bayes classifier having the assumption that the data from each label is drawn from a simple Gaussian distribution.

3.4 System Architecture

Having the best skilled and the most well-guarded applications are always at odds with one another. There is no way to achieve complete safety without some investment. In this case, the expenditures usually outnumber the time allotted for performance. The majority of applications vulnerable to SQL injection attacks are web applications. Here are several famous types of user-created SQL inputs that academics are experimenting with, as well as various methodologies and tools for detecting and avoiding such harmful inputs. When it comes to SQL injection attacks and application security, there are a number of challenges and issues that we face. In this project, we projected a unique framework using machine learning to detect SQLIAs. Instead of tackling the injection on the injection attacks on the server side by guarding the database, this model is designed to act as a filter on the client side, as shown in Figure 5.

3.5 System Components

The components of the system include the process at each level, program, utility, or part of a computer's operating system that helps to manage different areas of the system.

3.5.1 Dataset Pre-processing

Data preparation may be defined as the procedure that makes our dataset more appropriate for machine learning process. Pre-processing of data means change the dataset to lowercase, avoid unwanted spacing or blank space, etc. The disputation of clearing covariant and merit from statements like 1=1 and 2=0, that avoid mathematical statement symbol and change some possible trigger states in first approaching, which we later derelict in approval of the adjacent one, was only supported on dismissal of entropy from the trace. All the left functions, escape

symbols, and statements where then coded as a single value.

(1) Data Preparation: For the purpose of reducing data noise and improving precision, unnecessary spaces and escape sequences were eliminated and all the queries were converted into lowercase form.

(2) Training and Testing Datasets: The training and testing sets are randomly taken from the dataset with a conventional ratio of 70-30 (70% for training and 30% for testing) usingtrain_test_splitfunction built into sklearn library:

train_test_split(trainDF['text'], trainDF['label'], test_size = 0.3)

3.5.2 Parser

Our model came across a common adversity during the data processing phase where traditional machine learning model explicitly takes in structured tabular numeric data, but our collected data are entirely non-structured texts. This is where parser for text comes into play. Text parsing is the process of transforming given series of text into desired smaller components based on some specific rules. There are two common ways to parse texts: regular expression separation and tokenization. The former parses the targeted text using desired regular expressions such as "[a-z]" and "[\t]". The latter divides the text into tokens, where each token can be a character, a word or a phrase. In the case of SQLIAs, the regular expressions do not determine the malice of a query, the appropriate text parsing method for this model is tokenization. Queries are split into tokens of words.

3.5.3 Machine Learning Approach

Supervised learning method ology was chosen for the prese network as the best method to the trouble, that has some active models (SQL injection) and need a non-linear judgment-making. This method or learning algorithm take the data sample i.e., the training data and its associated output, labels or responses with each data samples during the training process.

The main objective of supervised learning algorithms is to learn an association between input data samples and corresponding outputs after performing multiple training data instances.

For example, we have **x**: Input variables and **Y**: Output variable.

Now, apply an algorithm to learn the mapping function from the input to output as follows:

$$Y=f(x) \qquad (2)$$

The main objective is to approximate the mapping function with sufficient accuracy so that, when new input data are presented, the corresponding output variables can be reliably predicted. In this study, a Gaussian Naïve Bayes (GNB) algorithm is employed in conjunction with a deep learning–based approach to detect SQLIAs.

3.5.4 NLP Techniques and Feature Extraction

For NLP, there are many featured engineering techniques, but the one that proves to be the most useful for this SQLIAs detecting model is Word Level Term Frequency–Inverse Document Frequency (TF-IDF) Vectors. TF-IDF stands for Term Frequency and Inverse Document Frequency, which is an important index for term searching and figuring out the relevancy of specific terms in a document. Term Frequency specifically compute how often a word occurs in a document, where Document Frequency determines how often a word happen in an entire set of documents. The most significant advantage of TF-IDF is that it will assume that the documents are just bags of words, where each word does not have any correlation to another. This method is simple but powerful for our use case since in SQL, there are no tense or grammar rules like human languages.

3.6 Building a Machine Learning Classifier in Python

Python is a popular object-oriented programing language having the capabilities of high-level programming language. It's easy to learn syntax and portability capability makes it popular these days. Python is the fifth most important language as well as most popular language for Machine learning and data science. The following are the features of Python that makes it the preferred choice of language for data science.

3.6.1 Components of Python Machine Learning Ecosystem

(1) Jupyter Notebook: Jupyter notebooks basically provides an interactive computational environment for developing Python based Data Science applications. They are formerly known as ipython notebooks.

(2) Scikit-learn: Another useful and most important python library for Data Science and machine learning in Python is Scikit-learn.

Let us begin by installing the Python module Scikit-learn, which is one of the most widely used and well-documented machine learning libraries for Python. Using a modified SQL Injection dataset, a Naïve Bayes classifier is employed to predict whether a given statement is an SQL injection statement or a non-SQL injection statement. Scikit-learn provides a comprehensive set of tools for building machine learning classifiers in Python. The steps for building a classifier in Python are as follows:

Step 1: Importing necessary python package

For building a classifier using scikit-learn, we need to import it. We can import it by using following script:

import sklearn

Step 2: Importing dataset

After importing necessary package, we need a dataset to build classification prediction model. We can import it from sklearn dataset or can use other one as per our requirement. We are going to use sklearn's Modified SQL Injection Dataset. We can import it with the help of following script:

from sklearn.datasets import load_modified_sql_injection

The following script will load the dataset:

data = load_modified_sql_injection()

We also need to organize the data and it can be done with the help of following scripts:

label_names = data['target_names']

labels = data['target']

feature_names = data['feature_names']

features = data['data']

The following command will print the name of the labels, SQL injection statement and non-SQL injection statement in case of our database.

print(label_names)

The output of the above command is the names of the labels:

['SQL injection statement', 'non-SQL injection statement']

These labels are mapped to binary values 0 and 1. SQL injection statement query is represented by 0 and non-SQL injection statement queryis represented by 1. The feature names and feature values of these labels can be seen with the help of following commands:

print(feature_names[0])

The output of the above command shows the names of the features corresponding to label 0, that is, the SQL injection statement class. Similarly, the feature names for the other label can be obtained using the following command:

print(feature_names[1])

Step3: Organizing data into training & testing sets

As we need to test our model on unseen data, we will divide our dataset into two parts**:** a training set and a test set. We can use train_test_split function of sklearnpython package to split the data into sets. The following command will import the function:

from sklearn.model_selection import train_test_split

Now, next command will split the data into training & testing data. In this example, we are using taking 30 percent of the data for testing purpose and 70 percent of the data for training purpose:

train, test, train_labels, test_labels = train_test_split(features,labels,test_size = 0.30, random_state = 42)

Step 4: Model evaluation

After dividing the data into training and testing we need to build the model. We will be using NB algorithm for this purpose. The following commands will import the module:

from sklearn.naive_bayes import GaussianNB

Then, initialize the model as follows:

gnb = GaussianNB()

Next, with the help of following command we can train the model:

model = gnb.fit(train, train_labels)

For evaluation purpose we need to make predictions. It can be done by using predict function as follows:

preds = gnb.predict(test)
print(preds)

Step 5: Finding accuracy

We can find the accuracy of the model build in previous step by comparing the two arrays namely test_labels and preds. We will be using the accuracy_score function to determine the accuracy.

from sklearn.metrics import accuracy_score
print(accuracy_score(test_labels,preds))
0.951754385965

Therefore, the above output shows that the accurate of NB classifier is 95.17%.

3.6.2 Optimization

Optimization is a critical component of our approach. When neural models of different architectures are evaluated under identical conditions (e.g., using the same dataset and the same number of training epochs), their performance can vary by as much as 95%. Manually selecting the optimal neural network architecture and training parameters is a stochastic and time-consuming process that often requires repeated experimentation and re-evaluation based on multiple factors. Optimization techniques enable systematic adjustment of these parameters and facilitate the selection of an effective configuration of model components without the need for extensive manual reconfiguration.

4. SYSTEM EVALUATION AND RESULT DISCUSSION

4.1 Introduction

This chapter shows in-depth evaluation details of the proposed system. In order to verify the effectiveness of the proposed algorithm, the results of SQL injection detection by traditional machine learning algorithms Support Vector Machine (SVM), K-nearest Neighbors (KNN) and Decision Tree (DT) and GNB algorithms are compared and analysed with the CNN algorithm proposed in this project. The algorithms developed in this project were developed using the Python programming language using the Jupyter Notebook and Scikit-learn libraries.

4.2 Evaluation Methodologies

4.2.1 Datasets

For the evaluation purpose we are using a dataset for training our model. The dataset in this project is from the dataset published on https://www.kaggle.com/sajid576/sqlinjection-dataset, which has a total of 30,919 data items and basically meets the experimental requirements. The training set is 70% of the dataset and the test set is 30%. The dataset is shown in Table 2.

4.2.2 Metrics

The fore most possibility is then assessed with the test dataset, and the factors of the algorithms that were used to compare. This result is obtained from calculating True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). From these values, the system calculates accuracy, precision, recall and F1-score at the last of the valuation procedure. This may be represented in a matrix form and known as confusion matrix. By checking the FN and FP values we decide which parameter is used to evaluate. F1-Score is used as the detection classifier performance in addition to detection accuracy (Accuracy), check-all rate (Recall), and check-accuracy rate (Precision). The F1-Scoreis used as a comprehensive evaluation criterion for classifier performance. TN presents the number of correctly predicted normal requests. TP presents the number of correctly predicted malicious requests. FN presents the number of incorrectly predicted normal requests, FP presents the number of incorrectly predicted malicious requests (Table 3).

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP'} \quad (3)$$

$$Precision = \frac{TP}{TP + FP'} \quad (4)$$

$$Recall = \frac{TP}{TP + FN'} \quad (5)$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (6)$$

The identification of effective detection mechanisms is critically important. Numerous approaches have been proposed to detect and mitigate SQLIAs; however, no single technique is capable of preventing all possible attack variants. Consequently, SQLIAs remain a major concern for cybersecurity professionals. Traditional signature-based detection systems have become insufficient, as attackers continuously introduce new and previously unseen SQLIA patterns. Therefore, modern SQLIA detection systems must be capable of identifying novel and unknown attacks.

Machine learning has increasingly been adopted in the cybersecurity domain to address such challenges. As this field is still evolving, a wide range of machine learning–specific libraries and open-source tools are available to support the detection and mitigation of security threats, including SQLIAs. In this study, the SQLIA detection problem is addressed using machine learning–based classification techniques. Incoming statements are classified as either SQLIA or non-SQLIA statements. Five algorithms are evaluated: CNN, GNB, SVM, KNN, and DT. Experimental

Table 2 Data Situation of SQL Injection Detection

Label	Description	Count	Ratio (%)
1	SQL Injection Statement	11330	36.64
0	Non-SQL Injection Statement	19589	63.36

Table 3 Confusion Matrix That Describes the Necessary Metrics to Evaluate the ML Classifiers (Jemal et al., 2020).

	Predicted as Normal Request	Predicted as Malicious Request
Normal Request	TN	FN
Malicious Request	FP	TP

Table 4 Model Comparison Results

Models	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
CNN	96	97	92	94
SVM	79	100	46	63
KNN	83	72	88	79
GNB	95	85	97	91
DT	87	67	98	80

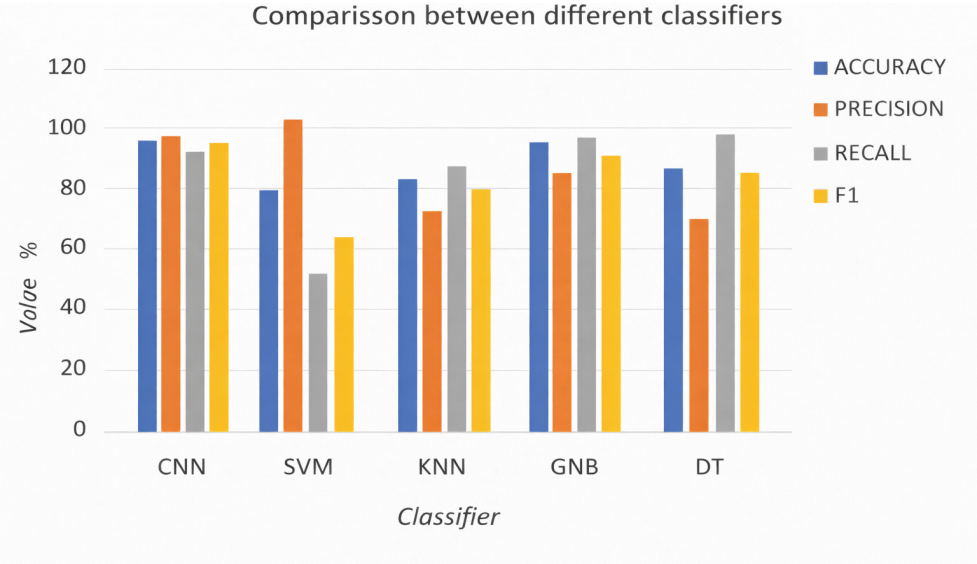

Figure 6 Comparison Between Different Classifiers

4.3 Results and Discussion

In this section, the experimental results are presented. Figure 6 illustrates the performance of the five classifiers evaluated in this study, namely CNN, GNB, SVM, KNN, and DT. The experiments were conducted using these algorithms with TF-IDF feature vectors at both the word level and n-gram level. Classification accuracy was measured for all models. A k-fold cross-validation method was employed to enhance the robustness and generalization capability of the models. The experiments were performed on a text classification task using datasets obtained from the Kaggle platform.

The results indicate that GNB outperforms other traditional machine learning algorithms in detecting SQL injection attacks. The GNB model achieved an accuracy of 95%. However, when compared with the deep learning approach reported by Wei et al. (2022) using the same Modified SQL Injection Dataset, the performance of GNB is slightly lower. Their CNN model achieved an accuracy of 96%, demonstrating the advantage of deep learning–based approaches for this task.

5. CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

An SQLIA on a web application poses a serious security threat, making the identification of effective detection mechanisms critically important. Numerous approaches have been proposed to detect and mitigate SQLIAs; however, no single technique can prevent all possible attack variants. As a result, SQLIAs remain a major concern for cybersecurity professionals. Traditional signature-based detection systems have

results show that GNB achieves an accuracy of 95%, while SVM, KNN, and DT achieve accuracy rates of 79%, 83%, and 87%, respectively.

Supervised machine learning approaches are generally capable of achieving higher accuracy by learning complex patterns from labeled data. Among all evaluated models, CNN was selected as the most effective approach for SQLIA classification. Through parameter tuning and optimization, the CNN model achieved an accuracy of 96%, outperforming traditional machine learning algorithms. Given that these results were obtained using an extensive dataset, it can be concluded that the CNN-based approach provides the most efficient solution for SQLIA detection in this study.

5.2 Recommendation

In future work, additional deep learning approaches and unsupervised ML algorithms will be investigated to further improve the performance metrics of SQLIA detection. Although CNN is adopted in this study to construct the detection model, other advanced algorithms can be explored and compared to identify the most effective model. Furthermore, feature reduction techniques will be applied to examine their impact on model performance, computational efficiency, and generalization capability. The dataset will also be expanded by incorporating additional categories of SQLIAs as new and emerging query patterns become available. This extension is expected to enhance the robustness and adaptability of the proposed detection framework.

REFERENCE

Abdalla Hadabi, A., Elsaman i, E., Abdallahy, A., & Elhabobz, R. (2022). An efficient model to detect and prevent SQL injection attack. *Journal of Karary University for Engineering and Science*, April 2022.

Akinsola, J. E. T., Awodele, O., Idowu, S. A., & Kuyoro, S. O. (2020). SQL injection attacks predictive analytics using supervised machine learning techniques. *International Journal of Computer Applications Technology and Research*, *9*(4), 139–149.

Alazab, A., & Khresiat, A. (2016). New strategy for mitigating SQL injection attack. *International Journal of Computer Applications*, *154*(11), 1–6.

Al-Maliki, M. H. A., & Jasim, M. N. (2022). Review of SQL injection attacks: Detection to enhance the security of the website from client-side attacks. *International Journal of Nonlinear Analysis and Applications*, *13*(1), 3773–3782. https://doi.org/10.22075/ijnaa.2022.6152

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaria, J., Fadhe, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Big Data Analytics*, *6*, 53. https://doi.org/10.1186/s40537-021-00444-8

AnandhaKrishnan, S. S., Sabu, A. N., Sajan, P. P., & Sreedeep, A. L. (2021). SQL injection detection using machine learning. *International Journal of Engineering Research & Technology*, *11*(3).

Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., & Nazario, J. (2007). Automated classification and analysis of internet malware. In *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, 178–197. Springer. https://doi.org/10.1007/978-3-540-74320-0_11

Chen, X. C. (2014). Research on algorithm and application of deep learning based on convolutional neural network (Master's thesis). Zhejiang University of Commerce and Industry, China.

Emmanuel, G., Joseph, S., Yakubu, J., & Abdulkadir, H. (2019). Performance evaluation of machine learning algorithms for detection and prevention of malware attacks. *IOSR Journal of Computer Engineering*, *21*(3), 18–27. https://doi.org/10.9790/0661-2103011827

Fang, J., Zhou, Y., Yu, Y., & Du, S. (2017). Fine-grained vehicle model recognition using a coarse-to-fine convolutional neural network architecture. *IEEE Transactions on Intelligent Transportation Systems*, *18*(7), 1782–1792. https://doi.org/10.1109/TITS.2016.2632682

Halfond, W. G. J., & Orso, A. (2005). AMNESIA: Analysis and monitoring for neutralizing SQL-injection attacks. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, 174–183. https://doi.org/10.1145/1101908.1101935

Holt, T. J., & Bossler, A. M. (2013). Examining the relationship between routine activities and malware infection indicators. *Journal of Contemporary Criminal Justice*, *29*(4), 420–436. https://doi.org/10.1177/1043986213507409

Indolia, S., Goswami, A. K., Mishra, S. P., & Asopa, P. (2018). Conceptual understanding of convolutional neural network: A deep learning approach. *Procedia Computer Science*, *132*, 679–688. https://doi.org/10.1016/j.procs.2018.05.069

Jemal, I., Cheikhrouhou, O., Hamam, H., & Mahfoudhi, A. (2020). SQL injection attack detection and prevention techniques using machine learning. *International Journal of Applied Engineering Research*, *15*(6), 569–580.

Joanna, H. B. J., Wafa, A. F. B. N., Nurrina, M. B. L., & Yu-Beng, L. (2022). SQL injection prevention in web application: A review. Faculty of Computing and Informatics, University Malaysia Sabah.

Joshi, A., & Geetha, V. (2014). SQL injection detection using machine learning. In *Proceedings of the International Conference on Control, Instrumentation, Communication and Computational Technologies*, 1111–1115. IEEE. https://doi.org/10.1109/ICCICCT.2014.6993128

Kamtuo, K., & Soomlek, C. (2016). Machine learning for SQL injection prevention on server-side scripting. In *Proceedings of the International Computer Science and Engineering Conference*, 1–6. IEEE. https://doi.org/10.1109/ICSEC.2016.7859877

Choudhary, K., Singh, A. K., & Gupta, R. (2016). A modified scheme for preventing web application against SQL injection attack. *International Journal of Computer Applications*, *141*(10), 27–31.

Lee, W. (2019). Malware and attack technologies knowledge area. Georgia Institute of Technology, The Cyber Security Body of Knowledge. https://www.cybok.org

McWhirter, P. R., Kifayat, K., Shi, Q., & Askwith, B. (2018). SQL injection attack classification through feature extraction of SQL query strings using a gap-weighted string subsequence kernel. *Journal of Information Security and Applications*, *40*, 199–216. https://doi.org/10.1016/j.jisa.2018.03.005

Palsson, F., Sveinsson, J. R., & Ulfarsson, M. O. (2017). Multispectral and hyperspectral image fusion using a 3-D convolutional neural network. *IEEE Geoscience and Remote Sensing Letters*, *14*(5), 639–643. https://doi.org/10.1109/LGRS.2017.2661929

Ross, K. (2018). SQL injection detection using machine learning techniques and multiple data sources (Master's thesis). San José State University.

Selvaganapathy, S., Nivaashini, M., & Natarajan, H. (2018). Deep belief network-based detection and categorization of malicious URLs. *Information Security Journal: A Global Perspective*, *27*(3), 145–161. https://doi.org/10.1080/19393555.2018.1456571

Sirinam, P., Imani, M., Juarez, M., & Wright, M. (2018). Deep fingerprinting: Undermining website fingerprinting defences with deep learning. In *Proceedings of the ACM SIGSAC Conference*

on Computer and Communications Security, 1928–1943. https://doi.org/10.1145/3243734.3243768

Stephens, N., Grosen, J., Salls, C., Dutcher, A., Wang, R., Corbetta, J., Shoshitaishvili, Y., Kruegel, C., & Vigna, G. (2016). Driller: Augmenting fuzzing through selective symbolic execution. In *Proceedings of the Network and Distributed System Security Symposium*. https://doi.org/10.14722/ndss.2016.23298

Wei, Z., Li, Y., Li, X., Shao, M., Mi, Y., Zhang, H., & Zhi, G. (2022). Deep neural network-based SQL injection detection method. *Security and Communication Networks*, 7094654. https://doi.org/10.1155/2022/7094654

Wu, X. R., & Chan, P. P. K. (2012). SQL injection attacks detection in adversarial environments by k-centers. In *Proceedings of the International Conference on Machine Learning and Cybernetics*, *1*, 406–410. IEEE. https://doi.org/10.1109/ICMLC.2012.6359493

Yuan, G., Li, B., Yao, Y., & Zhang, S. (2017). A deep learning enabled subspace spectral ensemble clustering approach for web anomaly detection. In *Proceedings of the International Joint Conference on Neural Networks*, 3896–3903. IEEE. https://doi.org/10.1109/IJCNN.2017.7966341

Zhou, Y., Wang, H., Xu, F., & Jin, Y. Q. (2016). Polarimetric SAR image classification using deep convolutional neural networks. *IEEE Geoscience and Remote Sensing Letters*, *13*(12), 1935–1939. https://doi.org/10.1109/LGRS.2016.2618840

Zhuo, Z., Cai, T., Zhang, X., & Lv, F. (2021). Long short-term memory on abstract syntax tree for SQL injection detection. *IET Software*, *15*(2), 188–197. https://doi.org/10.1049/SFW2.12018