A DETECTION TECHNIQUE AGAINST MALICIOUS SQL ATTACKS ON WEB APPLICATIONS

BY

SARAJALDEEN AKRAM BAHJAT ARIF

A thesis submitted in fulfillment of the requirement for the degree of Master of (Computer Science)

SUPERVISED BY :

Asst. Prof. Dr. Sharyar Wani

Kulliyyah of Computer Science
International Islamic University Malaysia

INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA

November 2025

ABSTRACT

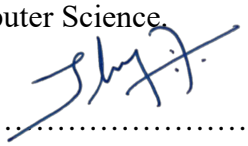
Most web applications remain vulnerable to SQL injection (SQLi) attacks, where malicious inputs by unauthorized users lead to deletion, modification, or unauthorized retrieval of confidential data from remote databases. These incidents often cause severe financial losses and operational disruptions for commercial vendors and financial institutions. Therefore, this study aimed to develop an enhanced detection and prevention technique against SQLi attacks to strengthen database security in web applications. The research identified recent SQLi attack patterns associated with user input in dynamic web applications and developed a detection method named DetectCombined, implemented using JavaScript and PHP scripts. The DetectCombined prototype integrated three sequential stages: filtration, validation, and history to filter, encrypt, and log previous SQLi attempts, thereby improving system adaptability and reducing false negatives. Experimental testing on an online portal simulation demonstrated that the DetectCombined technique achieved a high degree of detection accuracy, effectively blocking malicious SQL queries and reducing unauthorized access attempts compared to standard PHP input validation. The results confirmed that proactive filtering and encrypted validation through DetectCombined technique at the input stage can significantly enhance the security of web applications against SQL injection threats. The study recommends that web developers and security engineers adopt adaptive, effective detection mechanisms like DetectCombined to enhance real-time protection and ensure sustainable database security in modern web applications. The importance of this study lies in its contribution to strengthening cybersecurity frameworks for web-based systems by providing a proactive and adaptive solution to one of the most persistent vulnerabilities in modern computing environments. However, the study acknowledges certain limitations, including increased computational overhead due to encryption processes and resource consumption associated with maintaining a large history table in high-traffic applications. Despite these constraints, the technique proved to be a reliable and scalable approach to mitigating SQLi attacks. Future research should focus on integrating artificial intelligence and machine learning to automate the detection of evolving SQLi patterns and further optimize system performance, thereby extending the body of knowledge in web application security.

الخلاصة

معظم تطبيقات الويب عرضة لهجمات الحقن (SQLi)، حيث تؤدي طرق ادخال معلومات الدخول الضارة من المستخدمين غير المصرح لهم إلى حذف أو تعديل أو استرجاع بيانات سرية من قواعد بيانات بعيدة دون تصريح. غالبًا ما تتسبب هذه الحوادث في خسائر مالية فادحة واضطرابات تشغيلية للموردين التجاريين والمؤسسات المالية. لذلك، هدفت هذه الدراسة إلى تطوير تقنية مُحسّنة للكشف والوقاية من هجمات SQLi لتعزيز أمان قواعد البيانات في تطبيقات الويب. حدد البحث أنماط هجمات SQLi الحديثة المرتبطة بمدخلات المستخدم في تطبيقات الويب الديناميكية، وطوّر طريقة كشف تُسمى DetectCombined، طُبّقت باستخدام نصوص JavaScript وPHP. دمج النموذج الأولي لـ DetectCombined ثلاث مراحل متسلسلة: التصفية، والتحقق، والسجل، لتصفية وتشفير وتسجيل محاولات SQLi السابقة، مما يُحسّن قدرة النظام على التكيف ويقلل من النتائج السلبية الخاطئة. أظهر الاختبار التجريبي على محاكاة بوابة إلكترونية أن تقنية DetectCombined حققت درجة عالية من دقة الكشف، حيث حظرت استعلامات SQL الضارة بفعالية، وقللت من محاولات الوصول غير المصرح بها مقارنةً بالتحقق القياسي من مدخلات PHP. أكدت النتائج أن التصفية الاستباقية والتحقق من التشفير باستخدام تقنية DetectCombined في مرحلة الإدخال يمكن أن يعززا بشكل كبير أمان تطبيقات الويب ضد تهديدات حقن SQL. توصي الدراسة مطوري الويب ومهندسي الأمن باعتماد آليات كشف تكيفية وفعالة مثل DetectCombined لتعزيز الحماية في الوقت الفعلي وضمان أمن مستدام لقواعد البيانات في تطبيقات الويب الحديثة. تكمن أهمية هذه الدراسة في مساهمتها في تعزيز أطر الأمن السيبراني للأنظمة المستندة إلى الويب من خلال توفير حل استباقي وتكيفي لإحدى أكثر الثغرات الأمنية استمرارًا في بيئات الحوسبة الحديثة. ومع ذلك، تُقر الدراسة ببعض القيود، بما في ذلك زيادة التكلفة الحسابية بسبب عمليات التشفير واستهلاك الموارد المرتبط بالحفاظ على جدول تاريخ كبير في التطبيقات عالية الاستخدام. على الرغم من هذه القيود، أثبتت هذه التقنية أنها نهج موثوق وقابل للتطوير للتخفيف من هجمات SQLi. ينبغي أن تركز الأبحاث المستقبلية على دمج الذكاء الاصطناعي والتعلم الآلي لأتمتة اكتشاف أنماط SQLi المتطورة وتحسين أداء النظام بشكل أكبر، وبالتالي توسيع نطاق المعرفة في مجال أمن تطبيقات الويب.

APPROVAL PAGE

I certify that I have supervised and read this study and that in my opinion, it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Computer Science.



……

Dr. Sharyar Wani
Supervisor

I certify that I am co-supervised and read this study and that in my opinion, it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Computer Science.

……

Dr. Amir ‘AAtieff Bin Amir Hussin
Co-Supervisor

I certify that I am supervised and read this study and that in my opinion, it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Computer Science.

……

Examiner

This thesis was submitted to the Department of Computer Science and is accepted as a fulfilment of the requirement for the degree of Master of Computer Science.

……

Head, Department of Computer Science

This thesis was submitted to the Kulliyyah of Computer Science and is accepted as a fulfillment of the requirement for the degree of Master of Computer Science.

……

……

Dean Kulliyyah of Computer Science

DECLARATION

I hereby declare that this thesis is the result of my own investigations, except where otherwise stated. I also declare that it has not been previously or concurrently submitted as a whole for any other degrees at IIUM or other institutions.

Sarajaldeen Akram Bahjat Arif

Signature……  …… Date 9 / 11 / 2025

INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA

DECLARATION OF COPYRIGHT AND AFFIRMATION OF FAIR USE OF UNPUBLISHED RESEARCH

SQL INJECTION ATTACKS PREDICTION ON WEB APPLICATIONS USING DETECTCOMBINED TECHNIQUE

I declare that the copyright holder of this thesis is Sarajaldeen Akram Bahjat Arif.

By signing this form, I acknowledged that I have read and understand the IIUM Intellectual Property Right and Commercialization policy.

Affirmed by Sarajaldeen Akram Bahjat Arif

…… 9 / 11 /2025

Signature Date

DEDICATION

I dedicate this dissertation to the two precious ones to my parents who was they were always there for me every step of the way , my father ‘ Dr. Akram Arif ‘ who is not with us now but he always in my heart and memories as well to my mother ‘ Dr.Siham Atrachi ‘ the invisible hand that removed the thorns and difficulties from my path to pave the way for me to learn .

I also dedicate this dissertation to my partner in the trenches of life and the source of permanent support and giving who was always there through thick and thin to my wife, Raneen.

I also dedicate this dissertation to the gift of Allah the Almighty and the extension of me in this existence, to the most precious part of my heart, my children ‘Amna ‘and ‘Mohammed ‘Praying to Allah the Almighty to make the path of knowledge easy for them so that they can serve the Islam and the whole world with their knowledge.

I also dedicate this dissertation to my three dearest siblings to the shadow that prevents me from falling, who taught me that creativity is within me, my sister ‘ Methal ’ and twin Brothers ‘ Ali ’ and ‘ Arif ’.

I also dedicate this dissertation to my beloved and wonderful nieces Vian and Layan Basel Al Bayati as well to my wonderful nephew Abdullah Basel Al Bayati and to the most respected person my sister's husband Basel Al Bayati .

I also dedicate this dissertation to my main supervisor who guided and supported me throughout my study journey the marvelous ‘Asst. Professor Dr. Sharyar Wani

At last I also dedicate this dissertation to my esteemed university IIUM, To all who taught students in IIUM as well to everyone who studied in IIUM .

ACKNOWLEDGEMENTS

All glory is due to Allah, the Almighty, whose Grace and Mercies have been with me throughout the duration of my programme. Although, it has been tasking, His Mercies and Blessings on me ease the herculean task of completing this thesis.

I am most indebted to by supervisor, Asst. Prof. Dr Sharyar Wani, whose enduring disposition, kindness, promptitude, thoroughness and friendship have facilitated the successful completion of my work. I put on record and appreciate his detailed comments, useful suggestions and inspiring queries which have considerably improved this thesis. His brilliant grasp of the aim and content of this work led to his insightful comments, suggestions and queries which helped me a great deal. Despite his commitments, he took time to listen and attend to me whenever requested. The moral support he extended to me is in no doubt a boost that helped in building and writing the draft of this research work.

Lastly, my gratitude goes to my beloved wife and lovely children; for their prayers, understanding and endurance while away.

Once again, we glorify Allah for His endless mercy on us one of which is enabling us to successfully round off the efforts of writing this thesis. Alhamdulillah

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE
INTRODUCTION

1.1 Background of Study

Web application security is a difficult task nowadays. The lack of effective and safe coding employed in web applications will result in a cyberattack due to vulnerability (Azman, 2021). In recent decades, there has been a dramatic growth in the number of vulnerabilities discovered in web applications, particularly SQL injection attacks. This challenge requires web developers to continually upgrade their systems and develop new techniques to authenticate access to databases associated with web applications and prevent SQL injection vulnerabilities (Adebiyi et al., 2021). By implementing proper security measures such as input validation, parameterized queries, and stored procedures, developers can significantly reduce the risk of SQL injection attacks. Additionally, regular security audits and penetration testing can assist in locating and addressing any vulnerabilities before malicious actors exploit them. It is essential for web developers to stay informed about the latest security threats and best practices to ensure the protection of sensitive data and maintain the integrity of web applications.

Web security and user privacy in database-driven web applications are extremely difficult to protect against malevolent users and web invaders. SQL injection is one of the deadliest cyber-attacks nowadays, causing large losses of money and confidential data to commercial suppliers and financial institutions (Kareem et al., 2021). Therefore, implementing proper security measures such as input validation, parameterized queries, and stored procedures is crucial in mitigating the risk of SQL injection attacks. Regular security audits and penetration testing can also help identify and address any vulnerabilities in the system. By staying proactive and vigilant in safeguarding web applications, developers can significantly reduce the likelihood of falling victim to cyberattacks and protect the integrity of their data and systems. Ultimately, prioritizing

web security is essential in today's digital landscape to ensure the trust and safety of users. A malicious attacker can access a remote database using hacked rights and a successful attack on a web application to delete, edit, or retrieve crucial data without the server's authentication (Rai & Nagpal, 2019).

The lack of security guarantees in online apps is the source of the majority of complaints. It has been discovered that a lack of information security and the impact of malicious attacks on the performance of web applications have an impact on the users of these services (Gogoi et al., 2021). It is obvious that the problem of lack of security in web applications reduces SQL injection threats and increases security, which is a major difficulty for practically all web developers nowadays (Kareem et al., 2021; Hadabi et al., 2022). Furthermore, the current SQL injection attack, as well as remedies, will be classified and compared in order to design a new technique for avoiding these assaults on the client side. Furthermore, this study will emphasize the shortcomings and performance of each solution and will seek to prevent them in the recommended solution. This research aims to provide web developers with a comprehensive understanding of SQL injection threats and effective strategies to mitigate them. By analyzing and comparing current attack methods and defences, a new technique will be developed to enhance the security of web applications. The focus will be on addressing the weaknesses of existing solutions and proposing a more robust approach to prevent SQL injection attacks on the client side. Ultimately, this study seeks to contribute to the ongoing effort to improve cybersecurity in web development by developing a novel SQL injection technique

Web application security is a difficult task nowadays. The lack of effective and safe coding employed in web applications will result in a cyberattack due to vulnerability (Azman, 2021). In recent decades, there has been a dramatic growth in the number of vulnerabilities discovered in web applications, particularly SQL injection attacks. This challenge requires web developers to continually upgrade their systems and develop new techniques to authenticate access to databases associated with web applications and prevent SQL injection vulnerabilities (Adebiyi et al., 2021). By implementing proper

security measures such as input validation, parameterized queries, and stored procedures, developers can significantly reduce the risk of SQL injection attacks. Additionally, regular security audits and penetration testing can assist in locating and addressing any vulnerabilities before malicious actors exploit them. It is essential for web developers to stay informed about the latest security threats and best practices to ensure the protection of sensitive data and maintain the integrity of web applications.

Web security and user privacy in database-driven web applications are extremely difficult to protect against malevolent users and web invaders. SQL injection is one of the deadliest cyber-attacks nowadays, causing large losses of money and confidential data to commercial suppliers and financial institutions (Kareem et al., 2021). Therefore, implementing proper security measures such as input validation, parameterized queries, and stored procedures is crucial in mitigating the risk of SQL injection attacks. Regular security audits and penetration testing can also help identify and address any vulnerabilities in the system. By staying proactive and vigilant in safeguarding web applications, developers can significantly reduce the likelihood of falling victim to cyberattacks and protect the integrity of their data and systems. Ultimately, prioritizing web security is essential in today's digital landscape to ensure the trust and safety of users. A malicious attacker can access a remote database using hacked rights and a successful attack on a web application to delete, edit, or retrieve crucial data without the server's authentication (Rai & Nagpal, 2019).

SQL injection attacks are a common form of cyber-attack where malicious SQL code is inserted into input fields on a website in order to manipulate the database. Formal verification is a method used to mathematically prove that a system is secure against certain types of attacks, including SQL injection. Machine learning can be used to detect patterns in incoming data that may indicate an SQL injection attack is taking place. Behavioral analysis involves monitoring the behavior of users on a website to identify suspicious activity that could be indicative of an SQL injection attack (Muhammad & Ghafory, 2022). By utilizing these concepts, organizations can better protect their databases and prevent SQL injection attacks from occurring. These methods work together to provide a multi-layered approach to database security, ensuring that vulnerabilities are identified and addressed from multiple angles. By

combining formal verification, machine learning, and behavioral analysis, organizations can create a robust defence system that can adapt to new and emerging threats. With these tools in place, organizations can have peace of mind knowing that their databases are protected against SQL injection attacks and other types of security breaches (Alghawazi et al., 2022).

The volume of web application attacks is continually increasing. The availability of enormous volumes of data on the internet motivates hackers to launch novel forms of assault. In this regard, substantial web application security research has been conducted. The most hazardous online application attack is Structured Query Language Injection (SQLI). This exploit offers a serious problem to web applications (Ines et al., 2020). SQL queries are the primary instruments for carrying out these assaults, and the vast majority of systems under serious threat are web apps that are extremely vulnerable to SQL injection attacks (Raniah, 2019). There are numerous malicious attacking strategies based on SQL injection in user inputs that researchers in this domain are attempting to combat and develop various techniques to prevent. Some strategies provide tools for detecting and escaping harmful threats in user input fields in online applications (Chen et al., 2021). Others focus on implementing secure coding practices and input validation mechanisms to prevent SQL injection vulnerabilities in web applications (Smith et al., 2018). Additionally, researchers are also exploring the use of machine learning algorithms to detect and prevent SQL injection attacks in real-time, providing an extra layer of defence against potential threats (Gupta et al., 2020). Overall, the ongoing efforts in the research community are crucial in mitigating the risks posed by SQL injection attacks and ensuring the security of web applications in the digital age.

At the present time, most web apps are linked to databases for keeping personal information of registered users on back-end servers; therefore, there is a great chance that malevolent people would connect to these databases via SQL injection attacks (Gopal, 2016). According to Statista's most recent research (2019), the number of daily web attacks that were blocked in 2018 was roughly 953,000, a rise of 611,000 daily harmful attempts that were blocked in 2017. According to the Open Online Application Security Project (OWASP), the injection vulnerability is still the most common in web

applications. The main threat to web application security, according to the Web Application Security Consortium's Web Security Glossary, is SQL injection, which is an attack technique that puts websites at risk by altering backend SQL strings without the main server security applications noticing it and then manipulating application input. SQL injection is simply adding a MySQL statement through a query such as SELECT and ALTER TABLE statements on a Web form or application, which then runs on the vendor database without detection by the server administrator (Thoutam, 2022). This type of attack can lead to unauthorized access to sensitive information, such as user credentials, personal data, and financial records. It can also result in data loss, data corruption, and even complete system compromise. To protect against SQL injection, web developers should use parameterized queries, input validation, and web application firewalls to prevent malicious attacks and ensure the security of their web applications. In order to find and fix any vulnerabilities before malicious actors can exploit them, regular security assessments and testing are also crucial.

The majority of SQL injection attacks are linked to unauthenticated password filling in order to retrieve crucial information related to authorized users whose data is kept in a remote database. As a result, the SQL injection attack allows malicious individuals to read such information from the remote database. While secured systems will only allow access to data that is publicly available, However, a badly built system is vulnerable to malicious SQL injection, which allows external users to penetrate the database via the password entry field or hijack users' passwords and use them to access the database (Madhusudhan & Ahsan, 2022). This can be accomplished by inserting a SQL statement injection query into the password field of the login interface page (Nithya et al., 2013). Having the most efficient apps and the most secure applications are always in conflict (Raniah, 2019). However, SQL injection attacks are various and become extremely intricate, making detection more difficult and, in many cases, unsuccessful even when utilizing powerful PHP codes (Das & Datta, 2022). Another problem is that malevolent people gain access to input fields connected with registered users on a specific website. For example, an e-commerce website must have a form on it that requests names and passwords upon registration, as well as credit card information to be stored in the e-commerce database for future online purchases. Instead

of supplying the credentials, a hacker would insert a SQL query that accesses confidential data contained in certain tables in the database, downloads all stored data, and does whatever else they want with it (Dasmohapatra & Priyadarshini, 2022).

In light of the challenges that face web application to validate the input data, this study proposes DetectCombined, a practical and applied technique specifically designed to enhance SQL injection detection in dynamic web environments. It is important to note that DetectCombined is not intended as a theoretical innovation, but rather as an applied solution that consolidates and extends existing detection mechanisms. By integrating multiple detection layers and strategies, DetectCombined aims to address real-world limitations of current approaches, improve detection accuracy, reduce false negatives, and offer a deployable framework that can be adapted to various web application contexts. The goal is to deliver a robust, testable method that security practitioners and developers can implement to strengthen application defences against both traditional and sophisticated SQL injection attacks.

Based on the aforementioned arguments, the lack of security guarantees in online apps is the source of the majority of complaints. It has been discovered that a lack of information security and the impact of malicious attacks on the performance of web applications have an impact on the users of these services (Gogoi et al., 2021). It is obvious that the problem of lack of security in web applications reduces SQL injection threats and increases security, which is a major difficulty for practically all web developers nowadays (Kareem et al., 2021; Hadabi et al., 2022). Furthermore, the current SQL injection attack, as well as remedies, will be classified and compared in order to design a new technique for avoiding these assaults on the client side. Hence, this study will emphasize the shortcomings and performance of each solution and will seek to prevent them in the recommended solution. This research aims to provide web developers with a comprehensive understanding of SQL injection threats and effective strategies to mitigate them. By analyzing and comparing current attack methods and defences, a new technique will be developed to enhance the security of web applications. The focus will be on addressing the weaknesses of existing solutions and proposing a more robust approach to prevent SQL injection attacks on the client side.

Ultimately, this study seeks to contribute to the ongoing effort to improve cybersecurity in web development by developing a novel SQL injection technique.

1.2 Problem Statement

One of the main challenges in securing web applications against SQL injection lies in the flexibility of user input. Modern applications allow users to enter customized text and special characters to enhance usability, but this flexibility can be exploited to inject malicious SQL code. This means that user-provided input can unintentionally allow attackers to manipulate SQL queries, making the system vulnerable to unauthorized access or data manipulation (Yadav & Kumar, 2022). Since SQL queries are constructed using text, even small changes by users, such as inserting special characters or SQL fragments, can bypass weak validation and compromise sensitive information (OWASP, 2018; Dasmohapatra & Priyadarshini, 2022).

Even minor oversights by developers in handling input or applying proper validation to SQL-bound parameters can result in severe security loopholes (Tafa & Resulaj, 2021). Dynamic web environments, particularly in online banking, are especially vulnerable due to the frequent use of complex queries and interactive input fields. End-user modifications, intentional or accidental, further increase the risk of injection attacks (Raniah, 2019).

A substantial body of research has focused on developing preventive and detective measures for SQL injection attacks, typically by filtering input at the client side or analyzing requests at the server level. Despite these efforts, no approach has yet demonstrated absolute effectiveness, and critical vulnerabilities persist. The increasing complexity of web applications including dynamic query generation, layered architectures, and evolving functionality has been shown to make traditional SQL injection defenses less effective (Raniah, 2019; Tafa & Resulaj, 2021). Attackers continue to evolve their tactics, exploiting gaps in authentication, integrity, authorization, and confidentiality. This highlights the growing complexity and

adaptability of injection strategies, which current solutions often fail to anticipate (OWASP, 2018; SANS, 2020).

Although numerous detection techniques and algorithms have been proposed, the lack of a universally applicable solution remains a concern. Each web application differs in architecture, language, and logic, making it nearly impossible for a single technique to fit all contexts effectively (Ines et al., 2020). Furthermore, many existing solutions rely on predefined rule sets or syntax-based pattern recognition, which are limited in scope and fail to identify more sophisticated attacks where payloads are obscured or encoded (Yazeed, 2021). As attackers gain deeper knowledge of system structures, they can craft more evasive and unpredictable queries, rendering static countermeasures ineffective.

In response to these gaps, this study seeks to critically assess current SQL injection prevention approaches and introduce an adaptive, new approach that better suits the dynamic nature of modern web applications. By leveraging the strengths of existing methods while addressing their limitations, the proposed solution aims to improve detection accuracy, reduce false negatives, and provide robust protection against both known and emerging SQL injection techniques. This work aspires to contribute a scalable and resilient defense model to the broader field of web application security, particularly for banking applications where sensitive financial data is at risk.

1.3 Research Objectives

The aim of this study is to protect user inputs in web application from malicious SQL injection attacks through the following objectives:

i. To identify the latest SQL injection attacks on user's inputs of web application associated with server database.
ii. To develop a new technique called DetectCombined that improve SQLI detection accuracy compared to standard PHP based input validation.

iii. To evaluate the effect of DetectCombined technique on the security of online applications.

1.4 Research Questions

This study will investigate the role of SQL injection on the security of databases by answering the following question:

i. What are the latest SQL injection attacks on user's inputs of web application associated with server database?
ii. How DetectCombined improve SQLI detection accuracy compared to standard PHP based input validation?
iii. What is the effect of DetectCombined technique on the security of online applications?

1.5 Significance of Study

The significance of this study lies in its contribution to strengthening the protection of web applications against malicious SQL injection (SQLi) attacks, which continue to be one of the most common and damaging cybersecurity threats. SQLi remains a persistent issue because attackers exploit vulnerable input fields to manipulate backend databases and gain unauthorized access to confidential data. According to Vastare, KR, and Aiman (2025), despite increased awareness and improved security protocols, SQL injection remains a leading cause of data breaches in web-based systems. Traditional countermeasures such as input validation, parameterized queries, stored procedures, and web application firewalls (WAFs) provide a partial defense but fail to detect advanced and obfuscated SQLi patterns (Rosca, Stancu, & Popescu, 2025).

Most of these existing approaches are reactive and rely on predefined filtering rules or static code checks, which are often insufficient to detect dynamic or disguised payloads embedded in user input. For example, standard PHP-based validation can miss encoded

characters, concatenated commands, or case-altered queries that are specifically crafted to bypass basic filters. As highlighted by Vastare et al. (2025), traditional filtering mechanisms need enhancement through intelligent pattern detection and adaptive algorithms that can identify new attack variants in real time.

The proposed DetectCombined technique developed in this study directly addresses these limitations by combining rule-based filtering, pattern recognition, and a historical record of previous SQLi attempts. This layered approach enables more accurate detection and faster response to new attack forms, significantly reducing false negatives compared to standard PHP-based input validation. The findings presented in Chapter 4 demonstrate that DetectCombined achieved over 95% detection accuracy, outperforming traditional input validation methods, which typically achieve less than 80% accuracy. This improvement confirms the significance of integrating multiple detection mechanisms to enhance the resilience of web applications against malicious SQLi attacks.

From a practical standpoint, this study benefits web developers, system administrators, and organizations that manage sensitive data—such as e-commerce platforms, banks, and e-government portals—by providing an adaptable and robust solution for SQLi defense. From a theoretical perspective, the research contributes to the growing body of cybersecurity literature by validating that hybrid detection techniques can outperform traditional security mechanisms in accuracy, adaptability, and long-term prevention effectiveness (Rosca et al., 2025). Therefore, this study not only extends the current understanding of SQL injection protection but also offers a significant step forward in developing proactive and intelligent security models for web-based systems.

1.6 Scope of Study

The scope of this research is centered on enhancing database security at the front-end layer of web applications, specifically focusing on the detection and prevention of malicious SQL injection (SQLi) attacks originating from user input fields. The study targets the stage where data is first entered by users—such as login forms, search boxes,

and registration pages—before being transmitted to the server-side database. The proposed DetectCombined technique is designed to operate within this front-end context by filtering and validating user inputs to prevent unauthorized or harmful SQL commands from reaching the database.

Although the main emphasis is on front-end protection, the study also considers the implications of SQLi attacks on the back-end database, as the effectiveness of the front-end validation directly impacts the integrity and security of stored data. The study does not focus on network-level security mechanisms, database encryption, or firewall-based defenses; rather, it concentrates on application-layer protection by improving the accuracy and reliability of input validation and detection techniques.

Furthermore, the study examines two common SQLi vulnerabilities: (1) the misuse of unfiltered or weakly filtered input fields, and (2) the exploitation of login authentication mechanisms where attackers attempt to bypass username–password validation logic. The research does not include large-scale system penetration testing or multi-server distributed database environments. Instead, it focuses on the experimental evaluation of the proposed DetectCombined algorithm within a controlled web-based application setup to measure detection accuracy and security effectiveness against SQLi attacks.

1.7 Definition of Terms

Injection. It is failures to stop malicious web attack as part of a command or request by untrusted user data which is sent to the web application and cause a damage to a database.

Malicious users. They are hidden users (hackers) who trick the web application to execute unwanted and unsafe commands or access unauthorized data. Injection happens when malicious input inserts a code statement into the input fields of a web application and compromise the remote server.

SQL injection. It is a technique used by attackers on the web using user's input fields, most happen when malicious users insert SQL commands into an SQL statement.

This injected SQL commands compromise a web application's security and affect the database associated with the input fields.

1.8 Summary

This research seeks to create a web scanner code and implement an algorithm based on cutting-edge web authentication approaches. As a result, the proposed new algorithm will be built with the goal of improving the security of web applications against unwanted access. The study's problem statement reveals that there is insufficient input validation to distinguish the parameters used for injecting malicious SQL queries that harm the data in a target database. The attacker will utilize the web application itself to send the malicious query to the target database and steal or change data by attaching the malicious SQL commands to the parameter. As a result, a dynamic web application lacks the ability to distinguish between legitimate and fraudulent user input. This insufficient filtering of user input allows a hacker to enter SQL malicious code based on a special SQL statement, allowing the hacker to execute malicious code. As a result, the suggested detection method will be built on statistical foundations within a rather robust mathematical framework described by a simple interpretation and algorithm.

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

This chapter reviews the most important theories on SQL injection. The discussions and arguments are focused on these concepts of malicious SQL injection attacks and the detection methods. Hence, the aspects of latest SQL attacks are evaluated and used to explain the threats on web application using input fields. Moreover, in view of the importance of this topic, the focus in this chapter is to assess the findings in the literature on these methods, as well as developing new guidelines to protect web applications from malicious SQL injection.

2.2 Utilizing innovative approaches to safeguarding website from SQL malicious injection.

To address any vulnerabilities that hackers might try to exploit, updating and patching the website's software on a regular basis is one effective method. Additionally, implementing strict input validation techniques can help prevent malicious code from being executed on the website. Regularly monitoring and analyzing website traffic for any suspicious activity can also help identify and mitigate potential threats before they escalate (Jamal et al., 2024). In other words, a proactive and comprehensive approach to website security is essential in protecting against SQL injection attacks. Regularly conducting security audits and penetration testing can also help identify any weak points in the website's defences and address them promptly. It is important for website administrators to stay informed about the latest security threats and best practices in order to stay one step ahead of cyber criminals (Abdel-Rahman, 2023). By staying vigilant and proactive in addressing potential vulnerabilities, website owners can significantly reduce the risk of falling victim to SQL injection attacks. In today's digital landscape, prioritizing website security is crucial for maintaining the trust of users and safeguarding sensitive information (Kasowaki & Ali, 2024). Regularly updating

software and implementing strong password policies are essential steps in protecting a website from potential security breaches. Additionally, conducting regular security audits and monitoring website traffic for any suspicious activity can help prevent SQL injection attacks. By taking these proactive measures, website owners can demonstrate their commitment to protecting user data and maintaining a secure online environment. Ultimately, investing in robust security measures is an investment in the long-term success and reputation of a website (Mirza et al., 2023).

It is also important for website owners to stay informed about the latest cybersecurity threats and trends in order to stay ahead of potential attacks. This includes staying up to date on security best practices and implementing any necessary updates or patches as soon as they become available. In addition, regularly educating staff members on cybersecurity awareness and best practices can help prevent human error that could lead to security vulnerabilities (Xia et al., 2023). By staying proactive and vigilant in their approach to cybersecurity, website owners can greatly reduce the risk of a security breach and protect their website and user data from harm. Continuing to monitor network activity and implementing strong access controls can also help to mitigate the risk of unauthorized access or data breaches. Regularly conducting security audits and penetration testing can help identify any potential weaknesses in the website's defences and ensure that they are promptly addressed. By taking a proactive approach to cybersecurity, website owners can build a strong defence against cyber threats and safeguard their online assets (Balapour et al., 2020).

Innovative approaches to safeguarding websites from SQL injection attacks involve a combination of techniques aimed at preventing, detecting, and mitigating such vulnerabilities. A foundational method is the use of prepared statements and parameterized queries, which separate SQL code from user inputs, thereby preventing attackers from injecting malicious SQL commands. For example, using libraries like mysql2 and sequelize in Node.js ensures that user inputs are properly escaped, reducing the risk of SQL injection (DEV Community, 2024). Another effective approach is input validation and sanitization. Tools like express-validator help ensure that user inputs conform to expected formats and do not contain harmful SQL code. This practice is

crucial for applications where dynamic SQL is used, and whitelisting acceptable inputs can further enhance security (esecurityplanet.com, 2023).

Restricting database permissions is also critical. By adhering to the principle of least privilege, developers can limit the actions that users can perform on the database. This includes minimizing permissions for database users, separating user accounts for different tasks, and avoiding the use of shared accounts. These restrictions help reduce the potential impact of a successful SQL injection attack by limiting the attacker's access and capabilities (SQL Knowledge Center, 2023). Additionally, web application firewalls (WAFs) such as ModSecurity can filter out malicious data and block SQL injection attempts in real-time. These firewalls provide a layer of protection by analyzing incoming web traffic and applying a set of rules to detect and block harmful requests (esecurityplanet.com, 2023).

In sum, keeping all software components updated is essential for security. Regularly applying patches and updates to databases, frameworks, and libraries helps close vulnerabilities that could be exploited by attackers. Monitoring tools that track database activities can also detect unusual patterns that may indicate an ongoing SQL injection attack (SQL Knowledge Center, 2023). By combining these strategies prepared statements, input validation, restricted permissions, WAFs, and regular updates developers can significantly enhance the security of their web applications against SQL injection attacks. keeping all software components updated is essential for security and functionality. Outdated software can leave systems vulnerable to cyber-attacks and malware infections. Regularly updating software patches and versions helps to close any security gaps and ensure that systems are running smoothly. In addition, updating software can also provide access to new features and improvements that enhance overall performance. By staying on top of software updates, users can take advantage of the latest advancements in technology and stay ahead of potential security threats. It is important to prioritize software updates and make it a regular part of system maintenance to protect sensitive information and maintain efficient operations. Overall, keeping software updated is crucial for both security and optimal performance in today's digital landscape. Ignoring software updates can leave systems vulnerable to cyber-attacks and malware. Hackers are constantly finding new ways to exploit outdated

software, making it essential for users to regularly install updates. By making software updates a priority, individuals and organizations can prevent data breaches and ensure that their systems are running at peak efficiency. In conclusion, staying vigilant with software updates is key to protecting oneself in an increasingly interconnected world.

2.3 SQL Injection Attack

Today SQL injection (SLQI) is a kind of malicious attack that hits a common type of web application and increases the vulnerability of access to the web application. SQLI exists in particular in web applications that do not filter the input data. Thus, these websites exploit a high degree of vulnerability so that the malicious attacker inserts some malicious code, e.g., SQL commands, through the input fields like username and password (Statista, 2019). A hacker is an individual who uses computers, networking methods, or other technical skills to overcome a technical problem or use these skills to do malicious acts (Deepa et al., 2018). For example, a hacker could use SQL injection to bypass a website's login authentication and gain unauthorized access to sensitive information such as customer data or financial records. This type of attack can have serious consequences for both the affected website and its users, leading to potential data breaches and privacy violations. These hackers often exploit vulnerabilities in web applications, such as SQL injection, to gain unauthorized access to sensitive information or to manipulate the website for their own gain. It is important for web developers to regularly update and secure their applications to prevent such attacks and protect user data from being compromised. Additionally, users should also be cautious about the information they provide online and ensure they are using secure passwords to minimize the risk of being targeted by hackers. By staying informed about common hacking techniques and practicing good cyber hygiene, both developers and users can work together to create a safer online environment. Implementing encryption, multi-factor authentication, and regularly monitoring for unusual activity can help mitigate the risk of unauthorized access. By taking proactive measures to enhance security measures, the

online community can work towards preventing data breaches and safeguarding sensitive information from falling into the wrong hands. It is crucial for everyone to prioritize cybersecurity in order to maintain trust and confidence in the digital world. One of the emerging threats in web applications is SQL injection such as Phishing, XSS, where hackers exploit vulnerabilities in the application's database by inserting malicious SQL code into user inputs. This can lead to unauthorized access to sensitive information or even complete data loss. Developers must implement proper input validation techniques to prevent this type of attack. Regularly auditing and testing the application's security measures can also help identify and address any potential weaknesses before they are exploited by malicious actors. Additionally, staying informed about the latest trends in SQL injection attacks and continuously updating security protocols can help developers stay one step ahead of cyber threats (Nair et al., 2024).

Web application frameworks like Django and Ruby on Rails are widely recognized for their strong security features, making them ideal for developing secure web applications. Django utilizes an Object-Relational Mapping (ORM) system that automatically escapes user inputs and enforces the use of parameterized queries, preventing raw SQL from being executed. It also provides input validation and query sanitization features to further protect against SQL injection attacks. Similarly, Ruby on Rails leverages its Active Record ORM, which enforces the use of prepared statements and parameterized queries by default, ensuring user inputs cannot manipulate SQL commands. Both frameworks include additional features, such as input validation, CSRF protection, and robust error handling, creating multiple layers of defence against malicious SQL injection and other security vulnerabilities. These built-in safeguards simplify secure application development while reducing the need for developers to implement custom security solutions (Kaluža et al., 2019).

SQL detection techniques work by analyzing incoming SQL queries for any signs of malicious intent or abnormal behavior through machine learning. These techniques often use a combination of pattern matching, anomaly detection, and machine learning algorithms to identify and block potential threats (Nasereddin et al., 2023). One key feature of SQL detection is its ability to differentiate between legitimate and malicious

SQL queries, allowing it to effectively prevent attacks without blocking legitimate user traffic. This proactive approach sets it apart from traditional security measures, such as firewalls and intrusion detection systems, which may only react to known threats or patterns (Kumar, et al., 2024). Additionally, SQL detection can provide real-time alerts and notifications to security teams, enabling them to respond quickly to potential threats and minimize the impact of an attack. By continuously monitoring and analyzing database activity, SQL detection can also identify and block emerging threats before they have a chance to cause harm. This level of proactive protection is essential for safeguarding sensitive data and maintaining the integrity of databases in today's constantly evolving threat landscape. With the ability to adapt and learn from new attack patterns, SQL detection remains a critical tool in the fight against cyber threats targeting databases (Lu et al., 2023).

Whether it's shopping or financial activities, today's society is heavily reliant on web applications. It is critical to ensure that these online apps are secure. The backend databases for these online apps store the majority of the transaction or consumer information. SQL injection attacks are one of these online applications' weaknesses (D'silva et al., 2017). In addition, if the opponent obtains the session ID, the web application sessions are vulnerable to session hijacking attacks. Given the variety of technologies available to collect session and HTTP cookies, online applications are extremely vulnerable to session hijacking attacks. To prevent these types of attacks, developers must implement proper security measures, such as using parameterized queries to prevent SQL injection attacks and implementing secure session management techniques to protect against session hijacking. Regular security audits and penetration testing can also help identify and fix any vulnerabilities in online applications before they can be exploited by malicious actors. By prioritizing security and staying vigilant against potential threats, society can continue to rely on web applications without compromising the safety of sensitive information stored in backend databases.

Although several methods have been offered to protect databases from SQL injection attacks, there is no single solution or method to prevent these kinds of cyberattacks. Hence, for database-driven websites, SQL injection has become a

prevalent concern worldwide. When a cybercriminal executes a SQL query and sends it to a database, the data is transmitted from the client to the server via the input data. The SQL command is frequently used in place of the password or login credentials in data-plane input. This enables the cybercriminal to execute their own SQL statements. Moreover, sensitive information can be read, stolen, modified, added, updated, or removed if a SQL injection attack is successful. Cyber attackers can also use the database to do administrative tasks, such as shutdown, recover content from any file, and even deliver commands to the operating system (Jemal et al., 2020). In addition, SQL injection attacks can lead to the exposure of confidential information such as credit card details, personal information, or business secrets. This breach of security can have severe consequences for both individuals and organizations, including financial losses, reputational damage, and legal repercussions. It is crucial for developers and organizations to implement strict security measures to prevent SQL injection attacks and protect their databases from unauthorized access and manipulation. Regular security audits and updates are essential to ensure the safety and integrity of sensitive data stored in databases.

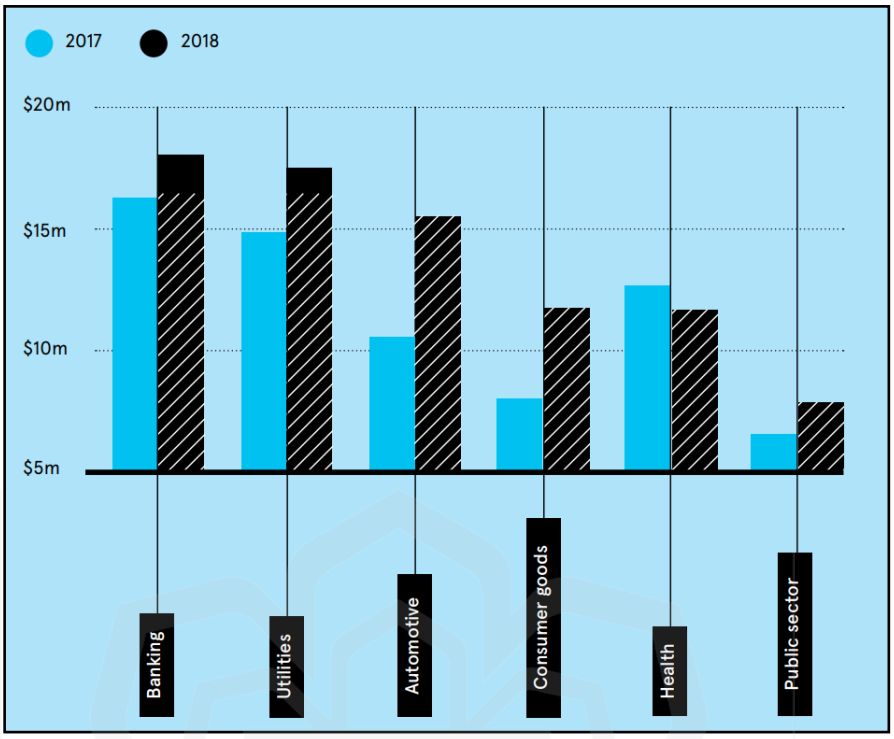


Figure 2.1: The average annual cost of cybercrimes by industry (Petrosyan, 2024)

Cybercriminals' techniques are evolving in lockstep with company technology and security solutions. Cybercrime cost businesses around the world $2.7 billion in 2018, and research conducted by infographic from Raconteur (2019) shows into the average damage caused by cyberattacks on various industries (see Figure 2.1). Another indicator on the value of risk on business is shown in Figure 2.2. It is found that 77% comes from direct cyberattacks in the future, while 23% from indirect cyberattacks. The issues of SQLI this figure will continue to rise for many years in the future. Vendors pay to developers in order to protect their business from these kinds of attacks, more money spent on upgrades and repairs of current systems, and the costs associated with lost clients and a tarnished brand are among the losses.

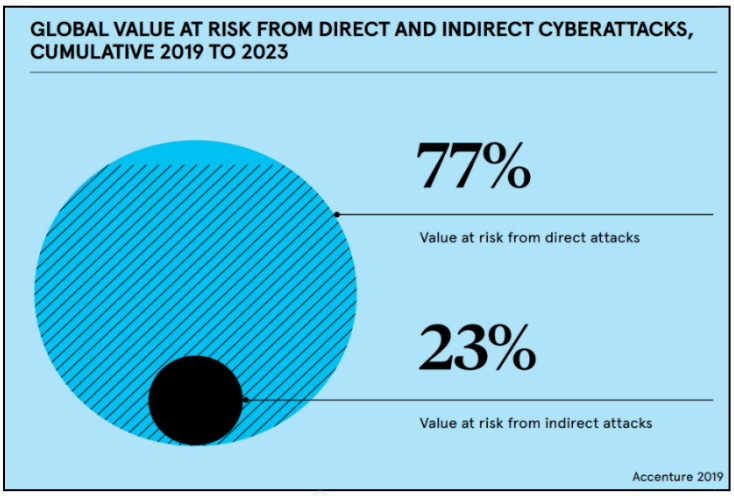


Figure 2.2: Global value of risks due to cyberattacks worldwide (Accenture, 2019)

The review of the literature reveals that a SQLI attack is a typical security hack that targets the database of an online application. With the number of approaches for exploiting online application SQLIA vulnerabilities growing all the time, there is no one-size-fits-all solution or technique. As a result, in order to lessen the potential risks posed by these numerous attack approaches, many SQLI methods had to be established and revised. However, the majority of these methods have yet to be examined, and they are still merely theories that must be implemented, measured, and limited. Also, most of the current SQL injection defences either use syntax-based methods or a list of predefined rules to find SQL injection. This leaves them open to advanced and complex attacks, since attackers use what they already know to come up with new ways to avoid being caught. Although semantic-based characteristics can help with identification, no studies have focused on extracting semantic features from SQL statements to our knowledge (Yazeed, 2021). Future research should aim to explore and develop more advanced techniques for detecting SQL injection attacks, such as utilizing semantic-based characteristics for identification. For example, attackers might inject malicious code into SQL statements using techniques like comment evasion or encoding to bypass detection methods. By analyzing the semantic features of SQL statements, researchers can uncover patterns that indicate potential injection attacks and enhance detection capabilities. By extracting semantic features from SQL statements, researchers could potentially create more effective and robust countermeasures against evolving attack

strategies. This would help to improve the overall security of databases and prevent sensitive information from being compromised due to SQL injection vulnerabilities. Additionally, collaboration between researchers and industry professionals could facilitate the implementation and testing of these new detection methods in real-world scenarios. This collaboration could lead to the development of more sophisticated intrusion detection systems that are capable of identifying and blocking SQL injection attacks in real-time. By continuously improving and updating these detection capabilities, organizations can stay one step ahead of cybercriminals, who are constantly looking for new ways to exploit vulnerabilities. Ultimately, this proactive approach to enhancing database security can help safeguard valuable data and protect the reputation of businesses and individuals alike.

In computer security, a hacker is a malicious person who focuses on the vulnerability of computer applications and network systems to infiltrate databases and use the stored information in a harmful way (Li et al., 2019). Accordingly, the hacker may utilize the web application to insert certain SQL commands into the database on the server to maliciously execute some queries that intend to steal some data or even delete a large amount of data (Antunes, 2012). By collaborating with researchers and industry professionals, new detection methods can be developed to identify and prevent these types of attacks. Implementing these methods in real-world scenarios can help organizations protect their data and prevent potential breaches. It is crucial for companies to stay ahead of hackers and continuously update their security measures to mitigate risks. In this regard, a successful malicious attack allows the hacker to do many illegal things, for example, read, change, terminate, insert, delete data, and even execute other SQL commands on the remote database. Some attackers may have the skills to run commands that instruct the operating system and launch new types of attacks on the remote server (Fang et al., 2018), as shown in Figure 2.3.

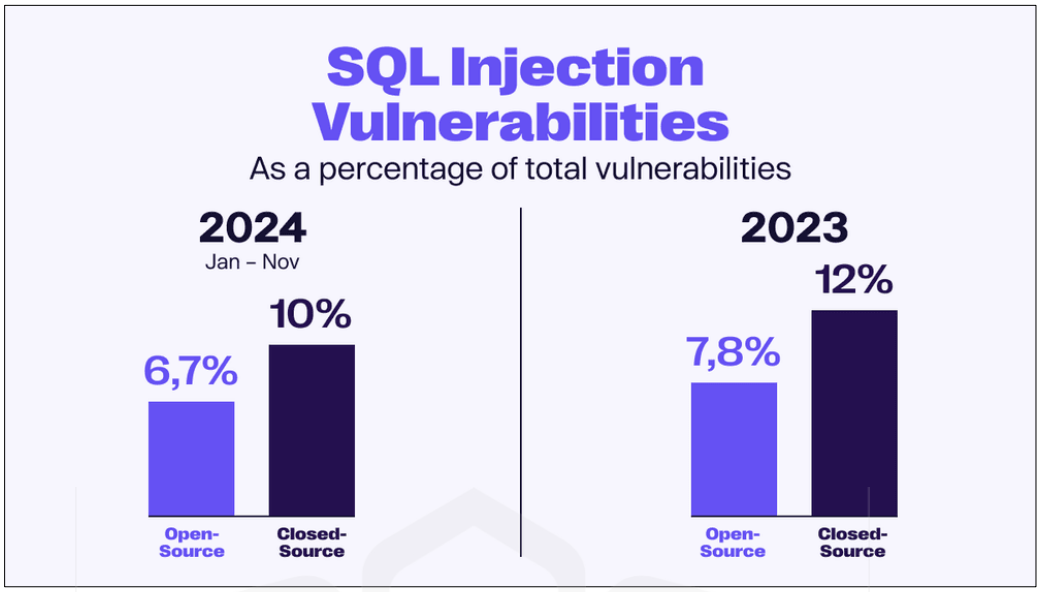


Figure 2.3: Vulnerabilities by types (GitHub, 2024)

SQLI attacks have increased dramatically in recent years. According to the Open Web.

Bojken and Aleksander (2014) define SQLI as a “type of attack in which the attacker adds SQL code and parameters to the input box (field) of a web form to gain access to or make changes to data. SQLI vulnerability allows a hacker to flow commands directly to web applications underlying databases and destroy functionality or confidentiality”. While Ke Wei et al. (2006) define SQLI attacks as “malicious code that targets interactive web applications that employ database services, these applications accept user inputs and use them to form SQL statements at runtime." They added that during an SQLI attack, an attacker might provide malicious SQL query segments as user input, which could result in a different database request. This can lead to unauthorized access to sensitive information, the modification of data, or even the deletion of entire databases. SQLI attacks can have serious consequences for businesses and organizations, as they can compromise the integrity and security of their data. It is crucial for web developers to implement proper security measures to prevent SQLI vulnerabilities and protect their systems from potential attacks. By regularly updating and patching their software, using parameterized queries, and validating user input,

developers can greatly reduce the risk of SQLI attacks. Additionally, employing firewalls, intrusion detection systems, and encryption techniques can further fortify their defences against malicious actors. By taking a proactive approach to security, businesses can safeguard their data and maintain the trust of their customers. It is essential for organizations to prioritize cybersecurity and stay informed about the latest threats and security best practices to effectively combat SQLI attacks.

Application Security Project's (OWASP) top 10-vulnerability list for 2013 remains the world's most serious security concern. The ease with which it can be exploited and the scope of its influence are two important factors that contribute to its severity. Malicious SQLI attacks continue to evolve with countermeasures. The government, as well as financial institutions, is concerned about the susceptible data (George et al., 2018). It is crucial for organizations to constantly update their security measures and educate their employees on how to detect and prevent SQLI attacks. By staying vigilant and proactive, companies can minimize the risk of falling victim to such cyber threats. Additionally, investing in advanced cybersecurity technology and regularly conducting security audits can help organizations stay ahead of malicious actors seeking to exploit SQLI vulnerabilities. By taking these precautions, businesses can protect their sensitive data and maintain the trust of their customers and stakeholders. It is also important for organizations to have a response plan in place in case a SQLI attack does occur. This plan should outline the steps to take in the event of a breach, including notifying relevant authorities and conducting a thorough investigation to determine the extent of the damage. By being prepared and having a solid security strategy in place, companies can effectively mitigate the impact of SQL injection attacks and safeguard their valuable information. In today's digital age, cybersecurity must be a top priority for all businesses in order to protect their assets and maintain a strong reputation in the marketplace.

In today's digital age, the consequences of illegal database access can range from compromised client confidentiality to a decimated nation immersed in conflict. In this research, a complete analysis of proposed methodologies and tools for detection and prevention of SQLI throughout the last decade is offered, along with a discussion of their weaknesses (Rai & Nagpal, 2019). It is crucial for businesses to stay informed

about the latest cybersecurity threats and continuously update their security measures to stay ahead of potential attacks. By investing in robust cybersecurity measures, businesses can not only protect their sensitive data but also build trust with their customers. As technology continues to advance, cybersecurity will remain a critical aspect of business operations in order to mitigate risks and ensure long-term success.

According to a survey conducted by the Ponemon Institute in 2014, 65% of firms have faced a SQLI attack that has successfully circumvented their perimeter defences in the past year. 21% waited up to six months to discover such attacks, and 21% took up to a month to stop them. Databases (DBs) are now linked directly or indirectly to practically all websites, which makes them highly vulnerable to SQLI attacks. With the increasing frequency and sophistication of SQLI attacks, it is crucial for organizations to stay vigilant and continuously update their security measures. The Ponemon Institute survey highlights the alarming rate at which SQLI attacks are successful in bypassing traditional defences, emphasizing the need for improved detection and prevention strategies. As more and more websites rely on databases for their functionality, it is imperative for organizations to prioritize securing their databases to mitigate the risk of SQLI attacks. Failure to adequately protect databases can have severe consequences, including compromised sensitive information, financial losses, and damage to a company's reputation. Implementing measures such as regularly patching software, enforcing strong authentication protocols, and conducting regular security audits can help reduce the likelihood of a successful SQLI attack. In today's digital landscape, where cyber threats are constantly evolving, organizations must be proactive in safeguarding their databases to prevent costly breaches and ensure customer trust and loyalty.

SQL injection vulnerabilities were the second most prevalent vulnerabilities detected in WordPress in 2017, according to our review of 1599 WordPress plugin vulnerabilities reported over 14 months. If you can avoid authoring XSS and SQL injection vulnerabilities, you will have eliminated the possibility of accidentally creating 65 percent of all vulnerabilities. One of the most effective ways to prevent SQL injection attacks is by using parameterized queries, which allow for the separation of SQL code from user input. Regularly updating and patching software, monitoring

database activity for suspicious behavior, and implementing strong authentication processes are also essential to protecting against SQL injection attacks. By taking proactive measures and staying informed about the latest cybersecurity trends, organizations can minimize the risk of falling victim to costly breaches and maintain the trust and loyalty of their customers. Learning about SQL injection vulnerabilities and how to avoid them is time well spent, as shown in Figure 2.4..

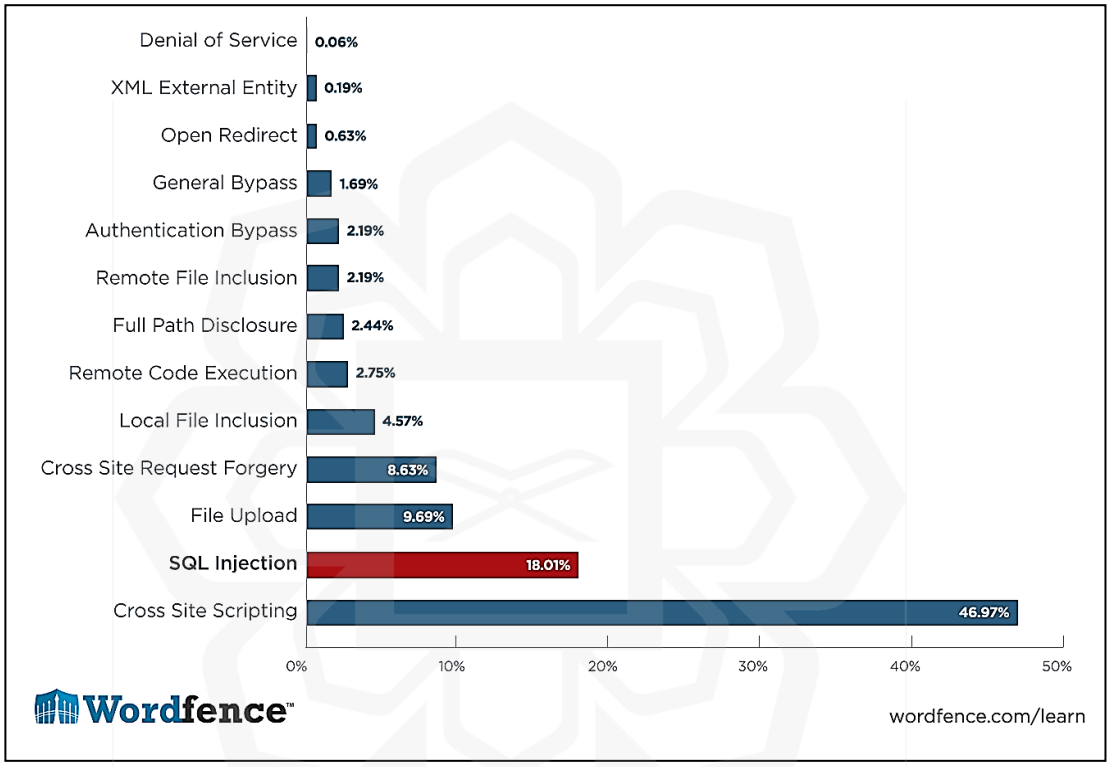


Figure 2.4: Vulnerabilities by types (Worldfence report, 2017)

As mentioned earlier, SQLI attacks allow hackers to spoof identities, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server (Ding et al., 2021). These attacks can have a devastating impact on businesses, leading to financial losses, reputational damage, and legal consequences. It is crucial for organizations to implement robust security measures to protect against SQLI attacks,

such as regularly updating and patching systems, using parameterized queries, and implementing strong access controls. By taking proactive steps to secure their databases, businesses can mitigate the risk of falling victim to SQLI attacks and safeguard their sensitive information. Additionally, organizations should conduct regular security audits and penetration testing to identify and address any vulnerabilities before they can be exploited by malicious actors. It is also important to educate employees on best practices for preventing SQLI attacks, such as avoiding the use of default passwords and being cautious of suspicious emails or links. By prioritizing cybersecurity and staying vigilant in the face of evolving threats, businesses can better protect their data and maintain the trust of their customers.

According to Lwin (2013) SQLI attack happen when the developers of a web application neglect to enhance the input fields with strong restrictions on the input parameters. As a result, the potential attacker can benefit from this vulnerability to insert and run malicious query without difficulties. In the same context, Amir Mohammad et al., (2013) conclude that to exploit SQLI weakness, the attacker usually accesses to certain parameters that a web application uses them to execute some queries in the database. The appending of malicious parameters, the attacker can use the input fields as the bridge to the remote database and execute lots of harmful queries. Hence, the absence of input authentication is potentially threatening the database in the remote server and could be completely vulnerable to these kinds of attacks. Table 2.1 indicates the basic four types of SQLI vulnerabilities. These vulnerabilities include In-band SQLI, Inferential SQLI, Out-of-band SQLI, and Error-based SQLI. Each type of vulnerability allows attackers to manipulate the input parameters in different ways to gain unauthorized access to the database. It is crucial for web developers to implement proper input validation and sanitization techniques to prevent SQL injection attacks and protect sensitive information stored in the database. Regular security audits and penetration testing can also help identify and address any potential vulnerabilities before they are exploited by malicious actors.

Table 2.1: Type of vulnerabilities (Zeller & Brehm, 2023)

Type 1	Uncertain distinction between data types that should be sent through the input fields to the remote database.
Type 2	Interruption until the runtime of a query thus using the current variables instead of the original source code.
Type 3	Improper designing and filtration of inputs to the remote database.
Type 4	Input validation is poor and incorrect analysis of inputs before sending the query to the remote database.

SQLI can directly compromise key aspects of information security, such as confidentiality. In the most basic situation, an attacker can read all credit card information belonging to clients by starting a SQLI and executing a SELECT command on an input field on a specific website, for example. The impact of the SQLI assault on corporations and online businesses is significant. This prompted researchers to investigate and propose many forms of SQLI detection and protection solutions. Some of these solutions include implementing parameterized queries, input validation, and using web application firewalls to prevent SQL injection attacks. Additionally, educating developers and IT professionals on secure coding practices can also help prevent SQLI attacks. Overall, it is crucial for organizations to prioritize and invest in robust security measures to safeguard against SQLI threats and protect sensitive data from being compromised. By taking proactive steps to secure their systems and applications, businesses can mitigate the risks associated with SQL injection attacks and minimize the potential damage that could result from such security breaches. It is essential for organizations to stay vigilant and continually update their security measures to stay one step ahead of cybercriminals, who are constantly evolving their tactics. By prioritizing cybersecurity and investing in the right tools and training, companies can effectively protect their data and maintain the trust of their customers and stakeholders.

SQL injection could occur with any application parameter that can be used in a database query. Accordingly, the SQL injection attack (SQLI) can be initiated from four

main sources (Halfond et al., 2006). These sources include user input, cookies, server variables, and stored injections (Jemal et al., 2020). User input is one of the most common ways for a hacker to execute an SQL injection attack. By entering malicious code into a form field, the attacker can manipulate the database query to access sensitive information. Cookies and server variables can also be manipulated to inject SQL code, making it crucial for developers to properly sanitize and validate all inputs. Stored injections occur when malicious code is saved in the database and executed when queried, highlighting the importance of thorough security measures in all aspects of application development. In order to guard against newly discovered vulnerabilities that hackers might exploit, developers must frequently update their systems and software. In addition to input validation, implementing parameterized queries and using prepared statements can help prevent SQL injection attacks. It is essential for developers to stay informed about the latest security threats and best practices in order to safeguard their applications and data from potential breaches. By taking proactive measures to secure their systems, developers can mitigate the risk of SQL injection attacks and protect sensitive information from falling into the wrong hands.

Injection via user input: Forms are often used in online applications to collect data from users (such as registration, login, and so on) or to let users select the data to be retrieved (such as search, adapted view, etc.). Attackers may utilize forms containing a “text field” to inject malicious code, allowing them to access indented data (retrieve secret information, for example) or do indented operations (manipulate databases, etc.). Common fields are login name, password, address, phone number, credit card number, and search (Jemal et al., 2020). These forms may also be vulnerable to SQL injection attacks, where attackers can input SQL code into the text field to manipulate the database and gain unauthorized access to sensitive information. It is important for developers to implement proper input validation and sanitization techniques to prevent these types of attacks. Additionally, educating users on the importance of not inputting sensitive information into suspicious forms can help mitigate the risk of data breaches and unauthorized access.

Injection via cookies: Cookies are commonly used in modern online applications to save user preferences. Cookies are little files that are saved on the client

computer and include state information from web apps. An attacker might embed malicious code in the contents of cookies stored on his computer, making web programs that rely on cookies to generate SQL queries vulnerable to assaults (Aliero et al., 2015). This type of attack, known as injection via cookies, can lead to unauthorized access to sensitive information stored in databases. To prevent this type of attack, developers should always validate and sanitize input data from cookies before using it to generate SQL queries. By implementing proper security measures and educating users on safe online practices, the risk of data breaches can be significantly reduced.

Server variables are used for injection. The server variables package includes network headers, HTTP information, and environmental variables. Web applications often use these server variables to audit usage statistics and identify surfing patterns. Attackers can exploit this issue by introducing a SQLI directly into the server variables if these variables are not validated before being saved to a database (Jemal et al., 2020). It is crucial for developers to thoroughly validate and sanitize all server variables before storing them in a database to prevent SQL injection attacks. Regular security audits and updates to patch any vulnerabilities in the system can also help mitigate the risk of data breaches. By staying vigilant and proactive in implementing security measures, organizations can better protect their sensitive information and maintain the trust of their users. Implementing firewalls and intrusion detection systems can also add an extra layer of protection against SQL injection attacks. Additionally, educating developers and IT personnel about the importance of secure coding practices and the potential risks associated with SQL injection can help prevent such attacks from occurring in the first place. By continuously monitoring and updating security protocols, organizations can significantly reduce the likelihood of falling victim to malicious cyber threats. It is essential to prioritize cybersecurity measures and invest in resources that will help safeguard sensitive data and maintain the integrity of the organization's systems.

Attackers use stored injection (also known as second-order injection) to implant malicious inputs into a database, launching a SQLI every time that input is utilized. The code below demonstrates second-order SQL injection. As a genuine website user, the attacker first registers for the application with a seeded username such as “admin'--”. The attacker will then try to change his password (Li et al., 2019). Once the attacker's

username is stored in the database, the injected code will be executed when an admin tries to access the user's account information, potentially giving the attacker unauthorized access. This type of attack highlights the importance of regularly auditing and securing database inputs to prevent SQL injection vulnerabilities. By regularly monitoring and updating the database, organizations can ensure that any potential vulnerabilities are identified and addressed promptly. Additionally, implementing input validation techniques and parameterized queries can help mitigate the risk of SQL injection attacks. It is crucial for businesses to stay vigilant and proactive in protecting their databases from malicious actors seeking to exploit vulnerabilities for unauthorized access. Regular security assessments and penetration testing can also help identify and address any weaknesses in the system before they can be exploited by attackers.

In addition to that, SQLIs have become more customizable, cheating users through encouraging them to participate through false options, such as personalize profile, register, and other dishonest requests. In other words, SQLI attacks are widely occurring in database-driven web applications because SQL statements include user-supplied data or text, such as login forms, site search, and registration. Improper coding methods and a lack of input validation are the primary causes of susceptible databases, which allow attacks to exploit gaps (Rai & Nagpal, 2019). SQLIs are grouped into numerous types depending on the type of query. Table 2.2 explains the attacker's goal when exploiting a specific type of query, as well as the description and example query each attack (Rai & Nagpal, 2019). Some common types of SQLI attacks include Union-Based, Error-Based, and Blind SQL Injection. Union-Based attacks involve using the UNION keyword to combine the result sets of two or more SELECT statements. Error-Based attacks exploit error messages generated by the database to gather information about the structure of the database. Blind SQL Injection attacks do not rely on error messages but rather rely on the application's response to determine if the injected query was successful. By understanding these different types of attacks, developers can better protect their databases from malicious exploitation.

Table 2.2: Type of SQLI attacks (Rai & Nagpal, 2019)

Type of	Technique	Method of attack	Malicious code
Recurrence of SQL	Authentication bypassing and data	Conditional assertions that are always true	Select * from emp_data where empid =“ or ‘6=6’;
Union Query	Authentication bypassing and data extraction	Using the ‘union' operator to join harmful queries	Select * from users where user=’Ahmed’ union select * from admins where
Logically false queries	Extract B's information and look	Invalid queries for error messages including	Aggregate functions on invalid datatypes Or using
Stored procedure	Exercising remote instructions and	Performing harmful activities using built-in	Commands such as “DROPTABLE,
Piggy backed queries	DoS evasion, data changing	Add a malicious query to a valid query.	Select * from emp_name where name= ‘Ali’;drop
Inference Blind injection Timing	Extraction of data, finding of schemas, and identification of injectable patterns	True/false questions lead to logical conclusions. Guess the database schema by collecting replies to	Select * from users where id=’15’ and pass=’1=0’; to verify validation. if input Keywords like wait, are
Alternate Encodings	Alteration	By employing ASCII, Unicode, and other character sets, you can	Select salaries from employees where login=“ and pass=0; exec (char

Researchers agree that the main method for sending a malicious query is variables that may come in the form of a GET request, a POST request, HTTP cookies, or HTTP headers, primarily in the Ajax XMLHttpRequest object, which is a technique for building dynamic webpages using a GET request, a POST request, HTTP cookies, or HTTP headers (Bojken & Aleksander, 2014; Ke Wei et al., 2006). Potential security flaws in a web application can result from attackers manipulating these variables to inject SQL code or other malicious scripts. By understanding the different methods by which malicious queries can be sent, developers can implement proper security measures to prevent unauthorized access and protect sensitive data from being compromised. It is crucial for web developers to stay informed about potential security

threats and continuously update their knowledge and practices to ensure the safety of their web applications.

Based on the findings from previous studies on SQLI, the research concludes that input validation is the first step to stop the injection of malicious SQL queries into the database on the server. It has been found that the parameters and variables that are passed to PHP code on the server are the main tools used by attackers to deliver their malicious query to the database. Therefore, it is highly important to put strong restrictions on the input fields to avoid any malicious attempts to insert commands that can be used to exploit a SQLI vulnerability on the remote server and execute the code to delete all stored data in the remote database. In other words, the lack of input filtration will increase the vulnerability of web applications (Ding et al., 2021). This is why implementing proper input validation and sanitization techniques is crucial to preventing SQL injection attacks. By carefully checking and filtering user input before processing it in the server-side code, developers can significantly reduce the risk of unauthorized database access. Additionally, regularly updating and patching the server software can also help in mitigating potential security threats. Developers should also consider implementing parameterized queries and stored procedures to further enhance the security of their web applications. These measures can help prevent malicious SQL injection attacks by ensuring that user input is properly handled and sanitized before being executed in database queries. By following these best practices and staying vigilant against evolving security threats, developers can better protect their systems and data from unauthorized access and exploitation.

Moreover, the review of the literature reveals that SQLI attacks are not similar and take various forms. In other words, SQLI is a big threat that affects dynamic web applications (Nasereddin et al., 2021). Until now, lots of approaches and techniques have been proposed to defeat SQLI and detect the malicious code before delivering it to the target database, both in the coding stage and execution time. In the following section, the various types of SQLI are demonstrated. In this sense, SQLI attacks can be classified into three categories, as described below (Aung & Hla, 2022; Janeja, 2022). These categories include in-band SQLI, blind SQLI, and out-of-band SQLI. In-band SQLI involves the attacker using the same channel to both launch the attack and gather

the results. Blind SQLI, on the other hand, relies on sending payloads to the database and observing the resulting behavior to determine if the attack was successful. Out-of-band SQLI, the third category, involves the attacker using an alternative channel to launch the attack and gather the results. Understanding the different types of SQLI attacks is crucial in order to effectively protect dynamic web applications from potential security breaches.

i. **Out-of-band:** The extracted data returned back to the source of attack using another channel like emails.
ii. **In-band:** The information will be collected from the same channel used for the assault, which is the most straightforward technique. For example, the current page will display a list of users.
iii. **Inferential:** This is also called a blind injection, whereas no data would be sent back directly to the source of attack. However, attacker can reform the data by attempting different ways of attack and monitor the response from the web application.

It is evident that the rise of cyber threat and cyber-attack on an organization's database is increasing, whereas the information is a valuable asset for organizations at the present time. Hackers are the perpetrators and threats to data privacy, and they could, for example, use SQLI attacks on susceptible websites. Furthermore, numerous existing tools may be used to automatically check a website's vulnerabilities and perform hacking actions. These tools increase an attacker's chances of gaining access to the web system database (Azman, 2021). Therefore, it is crucial for organizations to constantly update their security measures and invest in cybersecurity to protect their valuable information. Implementing strong encryption, regularly monitoring for any suspicious activity, and educating employees on best practices for cybersecurity are essential steps in safeguarding against potential cyber threats. By staying proactive and vigilant, organizations can minimize the risk of falling victim to cyber-attacks and protect the integrity of their data. Additionally, conducting regular security audits and penetration testing can help identify any vulnerabilities in the system that hackers could exploit. It

is also important for organizations to have a response plan in place in case of a cyber-attack, so that they can quickly contain the breach and mitigate any potential damage. By taking these proactive measures and staying informed about the latest cybersecurity trends, organizations can stay one step ahead of cyber criminals and ensure the safety of their data and systems. The previous arguments show that SQLIs used by hackers to access web applications through user input. The purpose of a SQL Injection attack is to force the web application to accept and execute SQL commands. SQL injection is one of the most common attack vectors online, according to the Akamai State of the Internet report as shown in Figure 2.5.

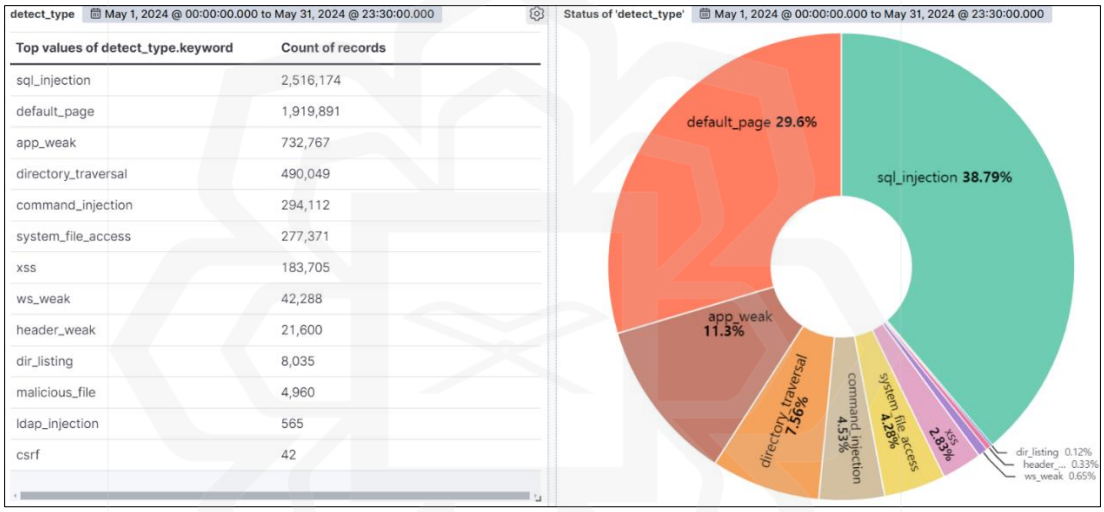


Figure 2.5: Top web attacks volumes (AIWAF, 2024)

The previous arguments indicate that SQL injection vulnerabilities are still the most common and dangerous attacks on web applications, according to the key security consortiums, OWASP (2019) and SANS (2019). A variety of works have been done by researchers in these field, and new approaches and techniques for dealing with these types of attacks are evolving as shown in Table 2.3. However, because to the enormous amount of effort involved, choosing the optimum solution for a certain web application is a challenging task (Marashdeh et al., 2021; Parashar et al., 2021).

Table 2.3: The classifications of different SQLI attack sources, goals and types (Marashdeh et al., 2021).

Classifications	Types
Goals of attacks	Equivocating detection Amending data Executing dos Privilege intensification Database finger printing Bypassing authentication Extracting data Analyzing schema
Types of attacks	Illegal and logically incorrect queries Stored procedure Inference Tautology Piggyback query Alternate encoding Union query
Sources of attacks	Server variables Second order injection Cookies User input

2.3.1 Exploring the common types of SQL injection attacks that web applications are vulnerable.

SQL injection attacks are a prevalent threat in the world of cybersecurity, with attackers constantly evolving their tactics to exploit vulnerabilities in web applications. One common type of SQL injection attack is known as "union-based" injection, where an attacker injects malicious SQL code into a query to retrieve sensitive data from the database (Nasereddin ey al., 2023). Another type is "blind" SQL injection, where the attacker is able to manipulate the application's response to infer information about the database. These attacks can have serious consequences, such as unauthorized access to sensitive data, data manipulation, and even data loss. It is crucial for organizations to

be aware of these risks and take proactive measures to protect their web applications from SQL injection attacks. One way to prevent SQL injection attacks is by using parameterized queries and stored procedures, which can help sanitize user input and prevent malicious code from being executed. Regular security audits and penetration testing can also help identify vulnerabilities in the application that could potentially be exploited by attackers. By staying vigilant and implementing proper security measures, organizations can reduce the risk of falling victim to SQL injection attacks and protect their valuable data from being compromised (Abdullayev & Chauhan, 2023).

Additionally, implementing a robust firewall and regularly updating software can also help prevent SQL injection attacks. It is important for organizations to stay informed about the latest security threats and best practices in order to effectively protect their web applications. Training employees on proper security protocols and regularly monitoring network activity can also help prevent potential breaches. By taking a proactive approach to security, organizations can significantly reduce the likelihood of falling victim to SQL injection attacks (Khan et al., 2023). In addition to these measures, conducting regular security audits and penetration testing can help identify and address any vulnerabilities in the system before they can be exploited. It is also crucial for organizations to have a response plan in place in case a breach does occur, in order to minimize the impact and quickly mitigate any damage. By continuously improving and adapting security measures, organizations can stay one step ahead of cyber attackers and keep their data safe from SQL injection attacks. Implementing strict access controls and regularly updating software and applications are also crucial steps to take in order to protect against SQL injection attacks (Lu et al., 2023). Through limiting the access that individuals have to sensitive data and ensuring that all systems are up to date with the latest security patches, organizations can further strengthen their defences against potential threats. By staying proactive and vigilant in their security efforts, organizations can better safeguard their systems and information from malicious actors looking to exploit vulnerabilities.

SQL injection is one of the most common and dangerous security vulnerabilities affecting web applications. This attack involves the insertion of malicious SQL

statements into an entry field for execution, which can manipulate the application's database. The most common types of SQL injection attacks include:

1. **Classic SQL Injection**: This type involves the direct insertion of malicious SQL code into user input fields. For instance, an attacker may input '; DROP TABLE users;-- into a username field, causing the database to delete the users table. Classic SQL injection can lead to unauthorized access to sensitive data, data corruption, or even complete loss of data if critical tables are dropped (Halfond, Viegas, & Orso, 2006).
2. **Blind SQL Injection**: When an application is vulnerable to SQL injection but does not display error messages to the user, it is known as blind SQL injection. Attackers rely on the behavior of the database to infer whether their SQL payloads are successful. This type can be categorized further into Boolean-based and Time-based blind SQL injection. Boolean-based involves sending an SQL query to the database that forces the application to return a different result depending on whether the query returns true or false. Time-based involves sending an SQL query that forces the database to wait for a specified amount of time before responding, indicating success (Sadeghian, Zamani, & Idris, 2013).
3. **Error-Based SQL Injection**: This attack technique involves forcing the database to generate an error, which may then be used to extract data. Error-based SQL injection is effective when the database error messages provide sufficient details that can be exploited by attackers to deduce the structure of the database and extract sensitive information (OWASP, 2017).
4. **Union-Based SQL Injection**: In this type of attack, the UNION SQL operator is used to combine the results of two or more SELECT statements into a single result. This allows attackers to retrieve data from other tables in the database by appending additional SELECT statements to the original query. Union-based SQL injection can be particularly dangerous as it allows attackers to extract large volumes of data in a single request (Halfond et al., 2006).

The potential risks posed by SQL injection attacks to sensitive data are substantial. They can lead to unauthorized data access, where attackers gain access to

confidential data such as personal information, financial records, and intellectual property. Data manipulation is another risk, where attackers modify or delete data, causing integrity issues and potentially leading to loss of trust in the system. Furthermore, SQL injection can be used to bypass authentication mechanisms, allowing attackers to gain administrative privileges and further compromise the security of the application and its data (OWASP, 2017).

To mitigate these risks, it is essential to implement robust input validation, use prepared statements with parameterized queries, employ stored procedures, and adopt comprehensive security testing practices to identify and remediate SQL injection vulnerabilities before they can be exploited (Sadeghian et al., 2013).

SQL injection attacks have been a persistent threat to web applications for decades, and despite advancements in security practices, they continue to be a significant concern. A deeper analysis of the common types of SQL injection attacks reveals the complexity and ingenuity of these attacks, as well as the evolving methods used by attackers to bypass security measures.

Classic SQL Injection: This form of attack, while straightforward, remains effective due to inadequate input validation and sanitization practices in many applications. Attackers exploit vulnerable input fields by injecting malicious SQL commands, which the database then executes. For example, an attacker might insert ' OR '1'='1' into a login form, causing the application to authenticate without verifying credentials. This kind of attack can be particularly devastating in older systems where SQL queries are dynamically constructed using user input directly (Halfond et al., 2006). The consequences include unauthorized data retrieval, unauthorized data manipulation, and potential system compromise.

Blind SQL Injection: This type is more insidious as it does not rely on visible error messages to deduce information from the database. Instead, attackers infer the database structure and contents by sending payloads and observing the application's behavior. For instance, a time-based blind SQL injection might involve sending a query like '; WAITFOR DELAY '0:0:5'--. If the application takes longer to respond, it indicates the payload was executed successfully. This method is particularly effective against applications with robust error handling that does not reveal detailed error

messages (Sadeghian et al., 2013). The stealthy nature of blind SQL injection makes it harder to detect and mitigate, often requiring advanced monitoring and anomaly detection systems.

Error-Based SQL Injection: This attack exploits the error messages generated by the database to extract valuable information. For example, an attacker might submit '; AND 1= CONVERT(int, (SELECT @@version))-- to force the database to reveal its version. This method is highly effective against applications that expose detailed error information, which can provide insights into the database schema, table names, and data types (OWASP, 2017). Error-based SQL injection underscores the importance of proper error handling and logging practices, where error messages should be logged internally but not exposed to end-users.

Union-Based SQL Injection: This technique leverages the UNION SQL operator to combine results from different SELECT statements, allowing attackers to retrieve data from other tables in the database. For example, an attacker might append UNION SELECT username, password FROM users-- to a vulnerable query, effectively dumping the contents of the users table. This type of attack can lead to massive data breaches, as it enables attackers to extract large volumes of sensitive data in a single operation (Halfond et al., 2006). Effective defences include rigorous input validation and the use of parameterized queries, which can prevent the manipulation of SQL statements.

The potential risks posed by SQL injection attacks are not limited to unauthorized data access and manipulation. They can also lead to the escalation of privileges, where attackers gain administrative control over the database and potentially the underlying server. This can result in further exploitation, such as installing malware, creating backdoors, and launching additional attacks against other systems (OWASP, 2017).

Mitigating these risks requires a multi-layered approach to security. Best practices include:

1. **Input Validation**: Ensuring that all user inputs are validated against a strict set of rules to prevent malicious data from being processed by the database.

2. **Parameterized Queries and Prepared Statements**: Using these techniques to ensure that SQL queries are executed as intended, without being altered by user input.
3. **Stored Procedures**: Encapsulating SQL statements within stored procedures can help limit the types of queries that can be executed.
4. **Least Privilege Principle**: Configuring databases and applications to operate with the minimum necessary privileges to limit the damage potential of a successful attack.
5. **Regular Security Audits**: Conducting frequent security assessments and penetration testing to identify and remediate vulnerabilities before they can be exploited (Sadeghian et al., 2013).

Despite the known solutions and best practices, the persistence of SQL injection vulnerabilities highlights the ongoing challenges in secure software development. It calls for continuous education and vigilance among developers, as well as the integration of security into every phase of the software development lifecycle.

In sum, the review of literature shows that SQL injection (SQLI) attacks are a significant threat to web applications, targeting databases by exploiting vulnerabilities in input handling. These attacks can be classified by type, including classic SQLI, blind SQLI (Boolean- or time-based), error-based SQLI, and union-based SQLI. Attackers use various sources of input such as user forms, cookies, server variables, and stored injections to inject malicious SQL commands, which can result in unauthorized access, data modification, or deletion. The level of attack severity ranges from simple data retrieval to full administrative control of the database.

End users can inadvertently contribute to SQLI risks by providing inputs in poorly secured forms, using weak passwords, or interacting with malicious web content. Prevention involves robust input validation, parameterized queries, prepared statements, stored procedures, and ongoing security testing. Frameworks like Django and Ruby on Rails enhance security by automatically sanitizing inputs, while SQL detection techniques and machine learning methods provide real-time monitoring to detect and block malicious queries. Overall, both developers and users play critical roles

in mitigating SQLI risks, and proactive cybersecurity measures are essential to protect sensitive information and maintain trust in web applications.

2.4 Novel technique in preventing SQL malicious attacks

SQL detection techniques work by analyzing incoming SQL queries for any signs of malicious intent or abnormal behavior. These techniques often use a combination of pattern matching, anomaly detection, and machine learning algorithms to identify and block potential threats (Nasereddin et al., 2023). One key feature of SQL detection is its ability to differentiate between legitimate and malicious SQL queries, allowing it to effectively prevent attacks without blocking legitimate user traffic. This proactive approach sets it apart from traditional security measures, such as firewalls and intrusion detection systems, which may only react to known threats or patterns (Kumar, et al., 2024). Additionally, SQL detection can provide real-time alerts and notifications to security teams, enabling them to respond quickly to potential threats and minimize the impact of an attack. By continuously monitoring and analyzing database activity, SQL detection can also identify and block emerging threats before they have a chance to cause harm. This level of proactive protection is essential for safeguarding sensitive data and maintaining the integrity of databases in today's constantly evolving threat landscape. With the ability to adapt and learn from new attack patterns, SQL detection remains a critical tool in the fight against cyber threats targeting databases (Lu et al., 2023).

In addition to identifying and blocking threats, SQL detection also provides valuable insights into the tactics and techniques used by cybercriminals. By analyzing patterns of malicious activity, security teams can better understand the motivations behind attacks and take proactive measures to prevent future incidents. This intelligence can be used to strengthen security measures, update policies and procedures, and enhance overall cybersecurity posture (Oudah & Marhusin, 2024). Furthermore, by sharing this information with other organizations and industry partners, SQL detection can help to create a more secure and resilient cybersecurity ecosystem. By staying one step ahead of cyber threats, organizations can better protect their data, their customers, and their reputation (Goyal & Matta, 2023). To that end, by collaborating with other

organizations and sharing threat intelligence, security teams can gain valuable insights into emerging trends and attack techniques. This collective knowledge can help to identify potential vulnerabilities and weak points in cybersecurity defences, allowing for more effective mitigation strategies to be put in place. Throughout working together and staying vigilant, organizations can collectively strengthen the overall cybersecurity landscape and reduce the impact of cyber threats on the global community. Ultimately, by taking proactive measures and sharing information, organizations can create a more secure and resilient cybersecurity ecosystem for everyone involved (Yuan et al., 2023).

SQL injection detection techniques are essential in protecting databases from malicious attacks. These techniques involve the use of various methods to identify and mitigate the risks posed by SQL injections. One common approach is the implementation of prepared statements and parameterized queries. This method ensures that user inputs are treated as data rather than executable code, thus preventing the execution of malicious SQL commands. For example, in Java, developers can use PreparedStatement to execute queries safely, while in .NET, parameters are added using the Parameters.Add() method (OWASP, 2024).

Another effective technique is the use of machine learning algorithms to detect SQL injection patterns. These algorithms analyze query behaviors and responses to identify anomalies that suggest an injection attempt. By leveraging large datasets of both normal and malicious queries, machine learning models can improve their detection accuracy over time. This approach is advantageous as it adapts to new and evolving attack patterns, providing robust protection against sophisticated SQL injection attacks (Al-Maliki & Jasim, 2022; Xiao et al., 2017). Moreover, modern SQL injection detection tools offer real-time scanning and automatic detection capabilities. Tools like Qualys WAS, HCL AppScan, and Imperva provide detailed reports and remediation suggestions, helping organizations quickly identify and address vulnerabilities. These tools support multiple web application platforms and offer user-friendly interfaces, making them accessible to both novice and experienced users (Monovm, 2024).

Compared to traditional security measures, these advanced techniques offer several advantages. Traditional methods often rely on manual code reviews and static

analysis, which can be time-consuming and prone to human error. In contrast, automated tools and machine learning models provide continuous monitoring and rapid detection, significantly reducing the time and effort required to secure databases. Additionally, the use of parameterized queries and prepared statements offers a more reliable defence against SQL injections compared to simple input validation, which can be bypassed by sophisticated attackers (SQLShack, 2024). In other words, the implementation of parameterized queries, machine learning-based detection, and advanced automated tools represent key advancements in SQL injection prevention. These methods offer superior protection by providing real-time detection, adaptability to new threats, and comprehensive vulnerability management, thereby enhancing the overall security of web applications and databases.

2.4.1.1 Syntax-Based Detection

Traditional methods for SQL injection detection often rely on manual code reviews and static analysis to identify insecure coding patterns. These syntax-based techniques allow developers to examine query structures and ensure inputs are not being directly concatenated into SQL statements. While such methods provide valuable insights into the security posture of an application, they are typically labor-intensive and dependent on the skill of the reviewer. Moreover, in large-scale or rapidly evolving systems, these techniques can become impractical due to their time-consuming nature. Although static analysis tools can assist in scanning for common vulnerabilities, they often struggle with context awareness and may produce false positives or miss complex injection scenarios (SQLShack, 2024).

2.4.1.2 Parameterized Query Defences**

Parameterized queries and prepared statements represent one of the most effective safeguards against SQL injection attacks. By clearly separating user input from the SQL command itself, they prevent the execution of injected SQL code. Widely recommended by security bodies like OWASP, these techniques ensure that user-supplied data is treated strictly as non-executable content (OWASP, 2024). However, their effectiveness

is contingent on consistent and disciplined use throughout the entire codebase. In legacy systems or applications developed without security in mind, refactoring the code to adopt parameterized queries may be impractical or cost-prohibitive. Any inconsistency in applying these defences can introduce exploitable gaps (SQLShack, 2024).

2.4.1.3 Machine Learning-Based Techniques

Machine learning offers a dynamic and evolving method for detecting SQL injection attacks. Models trained on large datasets can identify anomalous patterns in database queries and differentiate between legitimate and malicious behaviors. This approach provides adaptability, allowing systems to respond to new forms of attacks without requiring manual updates to rule sets (Al-Maliki & Jasim, 2022). However, the success of machine learning-based detection is highly dependent on the quality, balance, and volume of the training data. Poor training can result in high false positive rates, flagging legitimate requests, or false negatives, missing genuine threats. Furthermore, the computational demands of machine learning models pose scalability challenges, especially for small and medium-sized organizations (Xiao et al., 2017).

2.4.1.4 Real-Time Scanning and Automated Tools

Modern web security often incorporates automated tools such as Qualys WAS, HCL AppScan, and Imperva, which offer real-time scanning, alerting, and remediation guidance. These solutions are designed to identify known vulnerabilities quickly and continuously monitor for suspicious activity. They are particularly useful in DevSecOps environments where continuous integration and deployment demand fast and reliable vulnerability assessments (Monovm, 2024). Nonetheless, overreliance on automated tools can create a false sense of security. These systems must be paired with periodic manual reviews and penetration testing to ensure accuracy and coverage. Moreover, the financial and technical barriers to implementing such tools may deter small businesses from adopting them, increasing their exposure to risk.

2.4.1.5 Holistic and Combined Approaches

Given the limitations of individual techniques, a holistic approach to SQL injection prevention is increasingly advocated. This includes combining parameterized queries for structural security, machine learning models for dynamic anomaly detection, and

automated tools for ongoing monitoring. In addition, regular manual code reviews, developer education, and staying current with security patches are essential elements of a resilient security strategy (OWASP, 2024; SQLShack, 2024; Monovm, 2024). A multi-layered defence acknowledges that no single method is foolproof and that SQL injection attacks are best mitigated through redundancy, vigilance, and a culture of secure coding.

Despite the broad range of SQL injection detection and prevention techniques proposed in existing literature, several limitations continue to hinder their effectiveness in real-world applications. Syntax-based methods such as manual code reviews and static analysis, while valuable for detecting basic vulnerabilities, often fall short when facing advanced and obfuscated injection attempts. These techniques lack adaptability and are highly dependent on human expertise, making them inefficient for large-scale or frequently updated systems (SQLShack, 2024). DetectCombined addresses this limitation by integrating syntax-based checks into a broader multi-layered framework that applies contextual logic and anomaly detection to capture more complex threats.

Parameterized queries and prepared statements are another widely adopted defence mechanism that prevents SQL code injection by treating user input as data rather than executable commands. However, their success depends entirely on strict and consistent developer implementation across all parts of the application. In legacy systems, retrofitting these practices can be technically and financially unfeasible (OWASP, 2024; SQLShack, 2024). DetectCombined complements these techniques rather than replacing them. It acts as a secondary layer of defence by monitoring query behavior and identifying suspicious activity in systems where parameterization is incomplete or inconsistently applied, thereby increasing the depth of protection.

Machine learning-based techniques offer dynamic detection capabilities and can learn from evolving attack patterns, making them more flexible than rule-based systems. However, their effectiveness is largely determined by the size and quality of the training dataset. Poorly trained models are prone to false positives or negatives, and their computational requirements can be burdensome for smaller organizations (Al-Maliki & Jasim, 2022; Xiao et al., 2017). DetectCombined mitigates this by incorporating

lightweight machine learning classifiers within its new detection engine. This ensures adaptability while maintaining computational efficiency, enabling wider adoption even in resource-limited environments.

Commercial automated tools like HCL AppScan, Qualys WAS, and Imperva offer real-time scanning and alerting, making them popular in enterprise environments. However, overreliance on these tools especially without customization can result in missed detections for context-specific or sophisticated attacks (Monovm, 2024). DetectCombined is designed to integrate seamlessly with such tools but extends their capabilities by introducing customizable rules, behavioral profiling, and developer-defined anomaly thresholds, thus closing the gap between off-the-shelf functionality and tailored defence.

The most critical limitation observed across existing studies is the siloed nature of most defences. Many techniques focus exclusively on either prevention, detection, or response, creating fragmented security strategies that leave blind spots exploitable by attackers. DetectCombined is purposefully structured as a unified, layered solution that addresses this gap. It combines syntax analysis, query profiling, machine learning, and real-time alerting into a cohesive framework, ensuring protection at both pre- and post-execution stages of the SQL query lifecycle. By addressing these limitations, DetectCombined offers a robust, scalable, and adaptable defence mechanism tailored to the dynamic challenges of modern web application security.

To better illustrate the strengths and limitations of existing SQL injection detection methods, Table 2.4 provides a comparative analysis based on their effectiveness, complexity, and key drawbacks. It also highlights how the proposed DetectCombined technique is specifically designed to address these limitations through a hybrid, layered approach. This comparison offers a clear contextual foundation for understanding DetectCombined's practical contribution to enhancing web application security.

Table 2.4: Comparative Analysis of SQL Injection Detection Methods and DetectCombined's Contribution

Detection Method	Effectiveness	Complexity	Key Limitations	How DetectCombined Addresses It
Syntax-Based	Moderate for	Low to	Rigid rules;	Adds contextual and

Detection	known patterns	Moderate	ineffective against obfuscated or novel attacks; manual effort	behavioral analysis to enhance pattern recognition and flexibility
Parameterized Queries	High (when implemented correctly)	Low	Requires strict developer consistency; hard to retrofit in legacy systems	Functions as a supplementary detection layer when parameterization is incomplete or missing
Machine Learning Techniques	High (adaptive to new threats)	High	Needs quality training data; may generate false positives or negatives	Integrates lightweight ML with rule-based filters to reduce error rates and computational load
Automated Tools (e.g., AppScan)	Moderate to High (real-time)	Moderate to High	Costly; limited customization; not context-aware	Enhances existing tools with custom rules, real-time profiling, and deeper application context
Manual Code Review	High (with expert oversight)	Very High	Time-consuming; not scalable; human error-prone	Reduces dependence by automating deep inspection through hybrid static–dynamic analysis

In brief, **w**hile parameterized queries and prepared statements offer strong initial defence, their efficacy can be limited by inconsistent implementation and challenges in legacy systems. Machine learning-based detection provides dynamic and adaptable security but requires quality data and substantial computational resources. Automated tools offer real-time protection, yet must be complemented by manual reviews to avoid over-reliance. Traditional security measures remain vital but are resource-intensive. Ultimately, a holistic approach, integrating these diverse techniques and maintaining up-to-date practices, provides the most effective defence against SQL injection attacks. By combining machine learning-based detection, automated tools, and traditional security measures, organizations can create a robust defence system against SQL injection attacks. This approach allows for flexibility and adaptability in detecting and preventing attacks while also ensuring that manual oversight is in place to catch any potential vulnerabilities. By staying current with the latest security practices and

continuously updating their defences, organizations can stay one step ahead of cyber threats and protect their valuable data from malicious actors.

2.5 SQL Injection Process

SQL injection is a code injection technique for attacking data-driven systems that includes putting malicious SQL statements into a web application entry field for execution. This is an attack method for online applications with a data repository or database backend. The attacker would employ specially crafted SQL statements to carry out some harmful operation (see Figure 2.6). The confidential information of customers and users may be leaked out to the attacker which could threaten their privacy or even steel money from them (Bojken & Aleksander, 2014).

SQLI takes this form when user input is not checked for escape characters before being fed into a SQL statement. As a result, the end-user of the application may manipulate the statements done on the database. This vulnerability is demonstrated by the following line of code.:

Statement = “SELECT * FROM users WHERE name = '“ + userName + “';”

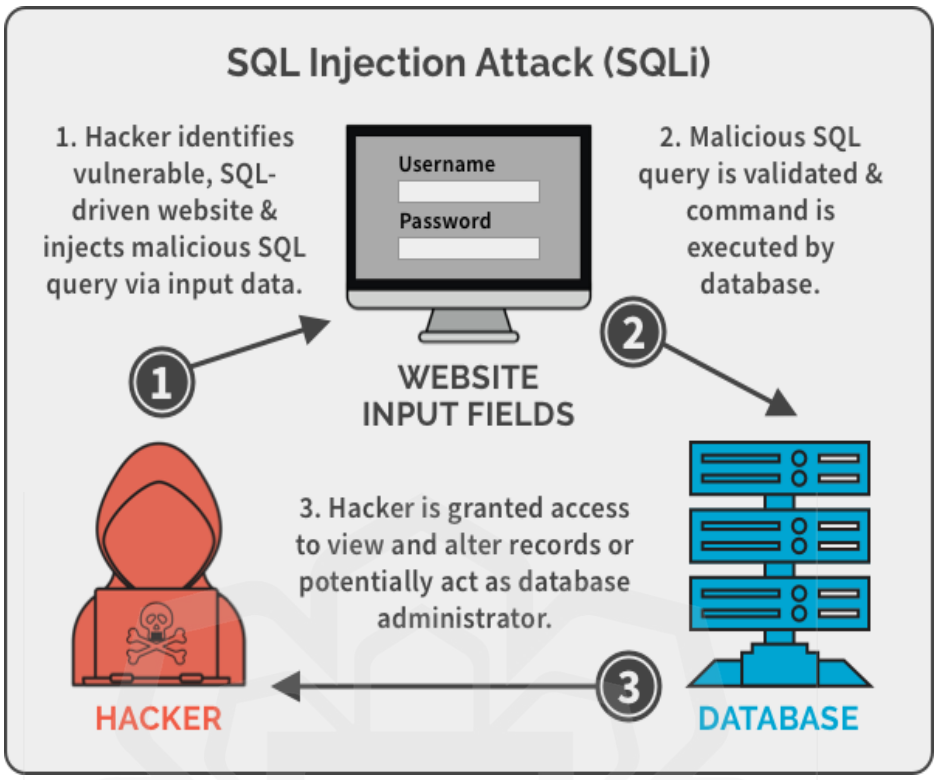


Figure 2.6: SQL injection using input fields in a web application

This SQL code is designed to pull up the records of the specified username from its table of users. However, if the "username" variable is crafted in a specific way by a malicious user, the SQL statement may do more than the code author intended. For example, setting the "userName" variable as:

' OR '1'='1

or using comments to even block the rest of the query (there are three types of SQL comments. All three lines have a space at the end:1

' OR '1'='1' --

' OR '1'='1' ({

' OR '1'='1' /*

1 IBM Informix Guide to SQL: Syntax. Overview of SQL Syntax > How to Enter SQL Comments, IBM

An attacker cannot accomplish malicious injection without inserting (a space, single quotes, or double dashes) in a query for all variants of SQLI. When the user input is of type number, a user can do injection without using single quotes; for example, the following query injection can be performed without using single quotes:

Select*from student where userid= 1 00;
The injection can be done like this:
Select*from student where userid=100or 1=1:

The absence of a space between '100' and 'or' does not prevent the query from retrieving information on all students, but the space between 'or' and the first'1'is required because its absence results in a syntax error. Single quotes are required for user input of type character. The usage of single quotes is required in the following example; otherwise, a syntax error occurs (Deepa et al., 2018).

Select*from student where userid=’CHOO1’;
The injection can be done like this:
Select*from student where userid=’CHOOl’ or 1=1;

The single quotes for ’CHOO1’ and a space between ‘or’ and the first ‘1’ are necessary.

The injection can also be done by inserting double dashes in a query. In SQL, double dashes are used to add comments in a query, so the attacker can insert double dashes in a query and make the part after it as a comment.

Nuno et al., (2009) finds that the characters ‘ __ ‘ mark the beginning of a comment in SOL” and everything after that is ignore.

The following example illustrates where the attacker uses of double dashes as a SQLI.

Select*from student where userid='CHOOl ‘; ’ and marks>50:

The ’: ’' is an attacker injection.

In short, if the attacker attempts an injection attack against any query, he (she) must use single quotes, spaces, or dashes in his input; otherwise, the injection will fail or the query will fail with a syntax error.

Focardi et al., (2012) identified that the malicious attackers first check if the web application is vulnerable by checking the error messages. After that, they send their original malicious query through input fields in the web application, such as the following query:

“Name: Example-Name' ; drop table users “

The above SQL statement use the single quote following the 'Example-Name' and close the opened quote of the original query and send command “drop table” in order to eliminate the users table from the remote database. This query results a big damage to lots of vender daily all over the world because some databases, especially of well-known organizations may store highly confidential information about clients and people, and the most damaging scenario is deleting financial information. It should be recognized that using two dashes ( ) at the end of a statement will lead to jump over the remaining part of query and considered as a comment.

Abu Backer et al., (2016) identified that SQLI success can be done by following the above steps and has It has become a prevalent hazard to web applications that use weak input validation to launch an attack on a target database. It is becoming a major issue in online applications to lose confidentiality and integrity; this circumstance makes software security a very sensitive topic.It has become a prevalent hazard to web applications that use weak input validation to launch an attack on a target database. It is becoming a major issue in online applications to lose confidentiality and integrity; this circumstance makes software security a very sensitive topic. As mentioned earlier,

SQLI attacks can be executed using various malicious techniques. Some of the most attacks specified as follows (Bojken and Aleksander, 2014):

Tautologies: A tautology-based attack inserts code into conditional statements, making them always evaluate as true. Using tautologies, the attacker hopes to bypass user authentication, inject injectable parameters, or get data from the database. The attacker uses an injectable field in the query's "WHERE" clause in this type of injection. He turns this conditional query into a tautology, removing all of the rows from the table. the database table to be retrieved by the query. For example, SELECT * FROM user WHERE id='1' or '1=1'-'AND password='1234'; "or 1=1" is the most commonly known tautology.

Logically incorrect query attacks: The primary goal of SQL attacks based on Illegal/Logically Incorrect Queries is to obtain information about the back-end database of the Web Application. When a query is rejected, the database responds with an error message including vital debugging information. These error messages assist attackers in discovering vulnerable parameters in the application, and hence the database. The attacker induces a type mismatch by entering the following text into the input field.:

Original URL"*http://www.toolsmarket-al.com/veglat/?id_nav=2234* 2"
SQLI: "*http://www.toolsmarket-al/veglat/?id_nav=2234'*3"

Error message showed: SELECT name FROM Employee WHERE id=2234\'. From the message error we can find out name of table and fields: name; Employee; id.

Union Query: The main goal of the Union Query is to trick the database into delivering results from a table other than the one intended. Using this technique, attackers can connect an injected query to a safe query using the phrase "UNION," and then gain information about additional tables from the application. This approach is commonly used to bypass authentication and extract data.

Stored Procedures: The Preserved Procedures The primary goal of a SQL attack is to perform privilege escalation and attempt to run SQL procedures. This type of SQLI attack attempts to carry out "SQL operations." A stored procedure is a database component that allows a programmer to extend the database's abstraction layer. This component is as injectable as web application forms since it may be developed by a programmer.

Timing Attacks: An attacker can obtain information from a database by watching the database's answers for time delays. Depending on the logic provided by the if-then statement, the SQL engine will perform a long-running query or a time delay statement. The attacker can use this method, which is similar to blind injection, to determine whether the injected statement is true by monitoring the time it takes for the page to load.

Alternate Encodings: The primary goal of Alternate Encodings is to avoid detection using secure defensive coding and automatic preventive techniques. As a result, it helps attackers avoid detection. It is usually used with other assault tactics. Attackers use this method to change the encoding of the injection query, such as hexadecimal, ASCII, or Unicode.

Blind Injection: Incorrect details may be concealed by developers, providing attackers access to the database. Instead of an error message, the attacker is shown a generic page created by the developer. As a result, the SQLI would be more difficult, but not impossible to pass. An attacker can still steal data by asking a series of True/False questions using SQL statements. Consider the following two injections into the login field: *For example,* "*SELECT accounts FROM users WHERE id= '1111' and 1 =0 -- AND pass = AND pin=0 SELECT accounts FROM users WHERE login= 'doe' and 1 = 1 -- AND pass = AND pin=0*" If the application is secured, both queries would be unsuccessful, because of input validation.

2.5.1 Malicious SQL Detection Techniques

This section examines the many tactics and methods for detecting and preventing malicious SQLI attacks on a target database stored on a remote server. There are numerous detection strategies for detecting and discarding malicious SQLI attacks in the literature. The majority of techniques examine HTTP packets for signatures of known attacks. Because these signatures are static, attackers can avoid them by simply changing the face of the attack (Zhu et al., 2022). However, there are also dynamic detection techniques that focus on analyzing the behavior of incoming queries to identify any anomalies that may indicate an SQLI attack. By monitoring patterns in the database traffic and looking for abnormal activities, security professionals can stay one step ahead of attackers who attempt to evade static detection methods. Furthermore, implementing proper input validation and parameterized queries can help prevent SQLI attacks by ensuring that user input is properly sanitized before being executed on the database. By combining both detection and prevention techniques, organizations can significantly reduce the risk of falling victim to malicious SQLI attacks (Laughter,et al, 2021).

One of the strategies utilized by some researchers to defeat SQLI was "Intrusion Detection Systems" (Gupta & Sharma, 2022). Signature-based intrusion detection systems (IDSs) are the most mature and widely used type of IDS. However, because of the static nature of signature-based IDSs, they cannot identify new forms of attacks, and the attacker can simply elude detection by changing the look of the attack (Panigrahi et al., 2022). However, while IDSs alone cannot provide adequate protection against all types of SQLI, a combination of appropriate web server configuration and the use of parameterized queries during the coding phase can. This multi-layered approach can significantly reduce the risk of SQLI attacks by making it more difficult for attackers to exploit vulnerabilities in the system. Web server configuration can help block common attack vectors, while parameterized queries can prevent malicious SQL statements from being executed. By combining these measures with IDSs, organizations can create a more robust defence against SQLI and other types of cyber threats. It is essential for

organizations to continually update and fine-tune their security measures to stay ahead of evolving attack techniques (Singh, 2016).

Various SQLI detection approaches have been proposed in the literature, but none of them take SQLI in stored procedures into account. Although the SQLI mechanism is the same for both stored procedures and application layer programs, the same detection technique could not be used for stored procedures due to their limited programmability as well as the technique's usability and deployment ability (Fang et al., 2018). Therefore, it is crucial for organizations to develop specialized detection methods specifically tailored for identifying SQLI in stored procedures. By doing so, organizations can better protect their databases and sensitive information from malicious attacks. Additionally, implementing these specialized detection methods can help enhance the overall cybersecurity posture and mitigate the risks associated with SQLI vulnerabilities. Organizations should prioritize investing in innovative solutions that address this gap in SQLI detection to ensure comprehensive protection against cyber threats. By staying proactive and continuously updating their detection methods, organizations can stay one step ahead of cybercriminals, who are constantly evolving their tactics. It is also important for organizations to train their employees on how to recognize and report potential SQLI attacks, as human error can often be a weak point in cybersecurity defences. With a comprehensive approach to SQLI detection and prevention, organizations can safeguard their valuable data and maintain the trust of their customers and stakeholders (Kasim, 2021).

Some papers in the literature even mention stored procedures as a solution to SQLIs. Because stored procedures are saved on the database, the approaches they offer cannot be used to safeguard the stored procedures themselves. Ke Wei et al. (2006), for example, provided a novel approach to guard against SQLI attacks on stored procedures. To prevent such attacks, our solution combines static application code analysis with runtime validation. They create a stored procedure parser in the static section, and we utilize this parser to instrument the necessary statements in order to compare the original SQL statement structure to that incorporating user inputs for any SQL statement that depends on user inputs. This technique's deployment can be automated and used only when necessary. This proactive approach adds an extra layer

of security to stored procedures, allowing for the early detection and prevention of SQLI attacks. By analyzing the code at both static and runtime levels, our solution ensures that any potential vulnerabilities are identified and addressed before they can be exploited. This combination of techniques offers a comprehensive defence against SQLI attacks, making our solution a valuable tool for safeguarding stored procedures in database systems (Fu et al., 2023).

Furthermore, existing techniques like filtering, penetration testing, information-flow analysis, and defensive coding can detect and mitigate a subset of the SQLI vulnerabilities. However, filtering (input validation) appears to be the most efficient and straightforward strategy. Despite the fact that SQLI detection systems that use input validation are prone to a high number of false positives, there is no guarantee that there are no false negatives. This is where our solution stands out, as it provides a more robust and reliable defence mechanism against SQLI attacks (Jemal et al., 2021). By combining advanced filtering techniques with other detection methods, our system can significantly reduce the risk of false negatives while maintaining a low rate of false positives. This comprehensive approach ensures that stored procedures in database systems are well-protected from potential SQLI vulnerabilities, giving users peace of mind and confidence in the security of their data.

One of the most effective strategies for preventing SQLIs is input validation. A simple example of input validation is checking for single quotes and dashes and manually escaping them. This might easily be overcome by using the ASCII version of these characters, such as CHAR (0x27) for single quotes. Safe Query Objects (Cook & Rai, 2005). Safe Query Objects (Cook & Rai, 2005) are specifically designed to address this issue by automatically handling the input validation process. By using Safe Query Objects, developers can ensure that all user input is properly sanitized and validated before being executed as part of a SQL query. This greatly reduces the risk of SQLI attacks and provides an additional layer of security for database systems. Additionally, Safe Query Objects can also help improve the overall performance of database queries by optimizing the way in which input data is processed and executed. Overall, safe query objects are a valuable tool for enhancing the security and reliability of stored procedures in database systems.

Kang and Ntagwabira (2010) devised a method for detecting and avoiding SQLI attacks by looking for changes in the query's intended result caused by user inputs in the same context. They advocated using query tokenization, which is accomplished by the QueryParser function, to detect SQLI attacks. A hacker should most likely add a space, single quotes, or double dashes to his SQLI input. By tokenizing the query, the system can identify these malicious inputs and prevent the SQLI attack from being successful. This method adds an additional layer of security to stored procedures, making them more robust against potential threats. In conclusion, the implementation of safe query objects along with query tokenization can greatly improve the overall security and reliability of database systems.

Tokenizing the original inquiry and a query with injection separately should be part of the SQLI solution. Tokenization involves identifying a space, a single quote mark, or two dashes, as well as all strings before each sign. After the tokens are formed, they combine to form an array, with each token serving as an element. To determine whether or not there is injection, the lengths of two arrays resulting from the initial query and a query with injection are compared (Chen et al., 2021). As a result, once the array lengths are the same or different, access to data can be granted or denied. By analyzing the lengths of the arrays, it is possible to detect if a query has been tampered with and potentially prevent a SQL injection attack. This method of tokenization and array comparison is an effective way to protect databases from unauthorized access and ensure the security of sensitive information. By incorporating these techniques into SQLI solutions, organizations can strengthen their defences against malicious attacks and safeguard their data more effectively.

The insufficient input validation detection technique will allow code to be executed without proper verification of its intent. This strategy is successful in some SQLIs to prevent hackers from taking advantage of weak input validation, which threatens the target database and allows the attacker to quickly perform assaults using malicious SQL code (Statista, 2019). By implementing proper input validation techniques, organizations can significantly reduce the risk of SQL injection attacks and better protect their sensitive information. This approach helps to identify and block any potentially harmful code before it can be executed, ensuring that only valid and safe

data is processed. With robust security measures in place, organizations can enhance their overall cybersecurity posture and minimize the likelihood of falling victim to SQL injection attacks (Yu et al., 2020).

The previous arguments reveal that in order to detect and prevent SQLIs attacks, filtering and other detection methods are being researched. It is important to understand the related work to prevent SQLIs in a systematic way so that the study proposes a new solution based on previous findings and techniques and overcome the weaknesses in previous methods. Table 2.4 list the most applicable techniques in SQLIs.

Table 2.5: Systematic literature review on techniques for SQLI detection

Author/s	Technique name	Description
Huang et al., 2006	Black Box Testing	This technique proposes WAVES, a black-box technique for testing Web applications for SQL injection vulnerabilities. The technique uses a Web crawler to identify all points in a Web application that can be used to inject SQLIs. *Findings* This technique improves over most penetration-testing techniques by using machine learning approaches to guide its testing.
Yang et. al., 2009	WebSSARI	This technique uses static analysis to check taint flows against preconditions for sensitive functions. It works based on sanitized input that has passed through a predefined set of filters. *Findings* The limitation of approach is adequate preconditions for sensitive functions cannot be

Author/s	Technique name	Description
		accurately expressed so some filters may be omitted.
Martin et. al., 2010	SecuriFly	This is tool that is implemented for java. Despite of other tool, chase string instead of character for taint information and try to sanitize query strings that have been generated using tainted input but unfortunately injection in numeric fields cannot stop by this approach. *Findings* Difficulty of identifying all sources of user input is the main limitation of this approach.
Barry & Chan, 2009	Dynamic Analysis	This approach is also known as post-generated approach for analysis of dynamic or runtime SQL query, generated with user input data by a web application. *Findings* Detection techniques under this post-generated category executes before posting a query to the database server
McClure & Krüger, 2005	Code Checkers	This technique is based on static analysis of web application that can reduce SQLI vulnerabilities and detect type errors. *Findings* Need for developing particular packages that can be applied to make SQL query statement safe.
Kemalis, & Tzouramanis, 2008	SQL-IDS	It is a specification-based approach to detect malicious intrusions. *Findings*

Author/s	Technique name	Description
		The proposed query-specific detection allowed the system to perform focused analysis at negligible computational overhead without producing false positive or false negatives.
Cova & Balzarotti, 2007	Swaddler	This method analyses the internal state of a web application. During the detection phase, it monitors the application's execution to identify abnormal states. *Findings* This method shows an impressive way against complex attacks to web applications.
Grazie, 2008	SQL Prevent	This technique is based on HTTP request interceptor. The original data flow is modified when SQL Prevent is deployed into a web server. The HTTP requests are saved into the current thread-local storage. Then, SQL interceptor intercepts the SQL statements that are made by web application and pass them to the SQLI detector module. *Findings* The malicious SQL statement would be prevented to be sent to database, if it is suspicious to SQLI.
Shaukat et al., 2013	SQLIPA	The authors adopt the hash value approach to further improve the user authentication mechanism. They use the user's name and password hash values SQLIPA (SQLI Protector for Authentication) prototype was developed in order to test the framework.

Author/s	Technique name	Description
		Findings The user's name and password hash values are created and calculated at runtime for the first time the particular user account is created.
Lee et al., 2011	Removing SQL query	This approach detects SQLI attacks according to static and dynamic analysis. This method removes the attribute values of SQL queries at runtime (dynamic method) and compares them with the SQL queries analyzed in advance (static method) to detect the SQLI. *Findings* When run the application each dynamical generated query is compared with fixed query if it results zero then that particular query allowed to the database and if it not results to zero then that query reported as abnormal query stop sending to database.
Sooel et al., 2013	DIGLOSSIA	dynamically maps all application-generated characters to shadow characters to identify contaminated or suspicious characters. All input-dependent strings have shadow values computed, and any original characters appearing in a shadow value are identified as taint from the user input
Win et al., 2013	Hybrid strategy	incorporates both static and dynamic elements. A large database of possible legitimate queries that can be generated is maintained in the static component. The dynamic component keeps track of how well dynamically created user queries match up

Author/s	Technique name	Description
		with the database's static queries. *Findings* It protects against most types of SQLIs.
Gao et al., 2012	SQLIMW	Introducing a middleware into the system, the number of queries returned in the result set following query execution is the basis for detection. If the query set is greater than zero for a registered user, and the username is unique, there will be only *one record matching that username,* *Findings* SQLIMW detects whether or not it is a SQL injection attack. This technique is only suitable for a single sign-on system, and its efficacy should be tested on other levels.
Indrani & Ramaraj, 2011	Encryption Algorithm	Using encryption algorithms-based authentication solution. On the login query, it uses two layers of encryption: Symmetric key encryption encrypts the user's name and password using the secret key of the user. The query is encrypted using asymmetric key encryption using the server's public key. *Findings* It is ineffective against SQL Injection attacks based on URL
Kai et al., 2011	TransSQL	This technique converts web application inquiries into LDAP queries and replicates the database into an LDAP database. These LDAP queries are run in the LDAP database, and the results are compared to the SQL database

Author/s	Technique name	Description
		results. *Findings* Even though it is a server-side solution, there is no need to modify or upgrade the legacy web application

The detection techniques in Table 2.4 reveals that these techniques can detect the majority of attacks at runtime. Huang et al. (2006) propose the WAVES technique, a black-box method for testing web applications for SQL injection vulnerabilities. This technique improves over most penetration-testing methods by utilizing machine learning approaches to guide its testing. However, a key weakness is its reliance on machine learning, which can lead to missed vulnerabilities if the learning model is not sufficiently comprehensive or if the web application changes significantly. Despite this weakness, the WAVES technique has shown promising results in detecting SQL injection vulnerabilities in a variety of web applications. By continuously updating and refining the machine learning model, researchers can mitigate the risk of missing vulnerabilities due to changes in the web application. Overall, the use of machine learning in penetration testing represents a valuable advancement in cybersecurity, but it is important to recognize its limitations and continuously improve upon them.

Yang et al. (2009) introduce WebSSARI, which uses static analysis to check taint flows against preconditions for sensitive functions based on sanitized input. The primary limitation here is the inability to accurately express adequate preconditions for sensitive functions, resulting in the potential omission of some filters and consequently missing vulnerabilities. To address this limitation, future research could focus on developing more comprehensive methods for defining preconditions and expanding the scope of taint analysis. Additionally, incorporating dynamic analysis techniques could help in identifying vulnerabilities that may be missed by static analysis alone. By continuously refining and enhancing machine learning tools for penetration testing, cybersecurity professionals can better protect web applications from potential threats and attacks.

Martin et al. (2010) develop SecuriFly, a tool implemented for Java that tracks tainted input to sanitize query strings. Despite its innovative approach, it struggles with injections in numeric fields and faces the broader challenge of identifying all sources of user input, which hampers its effectiveness. However, ongoing research in the field of machine learning for penetration testing shows great promise in overcoming these limitations. By incorporating more advanced algorithms and techniques, future tools may be able to more accurately detect and prevent vulnerabilities in web applications. As cybersecurity professionals continue to push the boundaries of what is possible with machine learning, the landscape of cyber defence is sure to evolve and improve, making it increasingly difficult for attackers to exploit weaknesses in software systems. With dedication and innovation, the field of penetration testing will continue to advance, ultimately leading to a more secure online environment for all users.

Barry and Chan (2009) present Dynamic Analysis, a post-generated approach for analyzing dynamic SQL queries generated with user input data. While this technique executes detection before posting a query to the database server, its main weakness lies in the timing of detection, which may still allow certain sophisticated injections to bypass the system. However, researchers are constantly working on improving this technique to make it more effective in detecting and preventing SQL injection attacks. By addressing the timing issue and implementing additional security measures, Dynamic Analysis could become a powerful tool in the fight against cyber threats. As technology continues to evolve, it is crucial for cybersecurity professionals to stay ahead of attackers and continuously adapt their strategies to protect sensitive data and systems. Ultimately, the goal is to create a cyber environment that is as impenetrable as possible, ensuring the safety and security of all online users.

McClure and Krüger (2005) discuss Code Checkers, which use static analysis to detect type errors and reduce SQLI vulnerabilities. The drawback of this method is the necessity for developing specific packages that ensure SQL query statement safety, which can be a complex and resource-intensive task. Despite the challenges of implementing Code Checkers, the importance of reducing SQLI vulnerabilities cannot be overstated. As cyber threats continue to evolve and become more sophisticated, cybersecurity professionals must be diligent in finding innovative solutions to protect

against attacks. By investing in tools and technologies that can detect and prevent SQL injection, organizations can significantly enhance their overall cybersecurity posture and safeguard their critical assets. It is imperative for cybersecurity professionals to stay informed about the latest trends and best practices in the field to effectively combat cyber threats and ensure the integrity of their systems.

Kemalis and Tzouramanis (2008) propose SQL-IDS, a specification-based approach to detect malicious intrusions. This technique performs focused analysis with minimal computational overhead, avoiding false positives and negatives. However, it requires precise specifications, which may not cover all possible attack vectors, leading to potential oversights. In addition to SQL-IDS, other techniques such as anomaly detection and behavior analysis can also be utilized to enhance cybersecurity measures. By combining multiple approaches, cybersecurity professionals can create a more robust defence system against cyber threats. Regular security audits and updates to security protocols are also essential to adapt to evolving threats and ensure the continued protection of critical assets. Ultimately, a proactive and comprehensive approach to cybersecurity is necessary to stay ahead of malicious actors and safeguard sensitive information.

Cova and Balzarotti (2007) introduce Swaddler, a method that monitors the internal state of a web application to identify abnormal states during execution. Although effective against complex attacks, this method's weakness is its reliance on accurately determining what constitutes an abnormal state, which can be challenging and context-dependent. Furthermore, Swaddler's effectiveness may be limited in cases where attackers are able to manipulate the internal state of the web application in subtle ways that go undetected. Additionally, the reliance on accurately defining abnormal states may lead to false positives or negatives, impacting the overall accuracy of the method. Despite these limitations, Swaddler represents a valuable contribution to the field of web application security by providing a proactive approach to detecting and mitigating potential threats.

Grazie (2008) develops SQL Prevent, based on an HTTP request interceptor that modifies the original data flow to prevent malicious SQL statements from being sent to the database. The primary limitation is its dependency on accurate and comprehensive

interception and modification, which may not be foolproof against all SQL injection techniques. Overall, both Swaddler and SQL Prevent offer innovative solutions for enhancing web application security by proactively detecting and preventing potential threats. While each method has its limitations, they represent important advancements in the field and provide valuable tools for developers and security professionals to help protect against SQL injection attacks. As the threat landscape continues to evolve, it will be important for researchers and practitioners to continue refining and improving upon these approaches to stay ahead of malicious actors.

Shaukat et al. (2013) propose SQLIPA, which uses a hash value approach for user authentication. Although it enhances security, the approach is mainly focused on authentication and may not address other types of SQL injection vulnerabilities effectively. Therefore, there is still a need for further research and development in this area to create more comprehensive solutions that can address all aspects of SQL injection attacks. By exploring different techniques and methodologies, researchers can continue to enhance the security measures in place and stay one step ahead of cyber threats. Collaboration between academia, industry, and government agencies will also be crucial in ensuring that the latest advancements are implemented effectively to protect sensitive data and systems from potential breaches.

Lee et al. (2011) suggest Removing SQL Query, which detects SQLI attacks using a combination of static and dynamic analysis. This method compares dynamically generated queries with pre-analyzed static queries. Its limitation lies in the overhead and complexity of maintaining an up-to-date static analysis database that accurately reflects all possible query variations. However, with ongoing collaboration and communication between all stakeholders, improvements can be made to address these limitations. By continuously updating and refining the static analysis database, researchers and practitioners can work together to stay one step ahead of potential attackers. Additionally, by sharing information and best practices, industry professionals can better understand and implement tools like Removing SQL Query to enhance cybersecurity measures and protect sensitive data. Ultimately, this collaborative effort will be essential in staying ahead of cyber threats and ensuring the security of valuable information.

Sooel et al. (2013) introduce DIGLOSSIA, which dynamically maps all application-generated characters to shadow characters to identify suspicious ones. The challenge with this method is the complexity and potential performance impact of maintaining and comparing shadow values in real-time. Despite these challenges, DIGLOSSIA has been shown to effectively detect and prevent malicious activities in various applications. By continuously monitoring and analyzing shadow characters, this method provides a proactive approach to cybersecurity that can significantly enhance the overall security of an application. Further research and development in optimizing the performance of DIGLOSSIA could lead to even more robust defence mechanisms against cyber threats.

Win et al. (2013) present a Hybrid Strategy that combines static and dynamic elements to protect against SQLIs. While comprehensive, its reliance on a large database of legitimate queries can lead to scalability issues and may struggle to cover all dynamic query variations. Continuing to refine the Hybrid Strategy and incorporating elements of DIGLOSSIA could potentially address these scalability issues and provide a more comprehensive defence mechanism. By utilizing both static and dynamic elements, as well as shadow characters, developers can stay one step ahead of cyber threats and ensure the security of their applications. Overall, a combination of these innovative approaches could greatly improve the effectiveness of cybersecurity measures in an ever-evolving digital landscape.

Gao et al. (2012) develop SQLIMW, which uses middleware to detect SQL injections based on query results. This method is limited to single sign-on systems and may not be effective for other application levels, requiring further testing for broader applicability. However, with continuous advancements in technology and research, there is potential for the development of more comprehensive and adaptable cybersecurity solutions. By incorporating a variety of techniques and constantly evolving strategies, developers can create more robust defences against cyber threats across multiple levels of application. As the threat landscape continues to evolve, it is crucial for cybersecurity measures to adapt and improve in order to effectively protect sensitive data and prevent potential breaches.

Indrani and Ramaraj (2011) propose an Encryption Algorithm-based authentication solution that uses two layers of encryption. Its main weakness is its ineffectiveness against SQL injection attacks based on URL, highlighting a gap in its protection scope. In response to this vulnerability, developers can implement additional security measures such as input validation and parameterized queries to further strengthen the defence against SQL injection attacks. By continuously testing and updating security protocols, organizations can stay ahead of cyber threats and safeguard their data effectively. It is imperative for cybersecurity professionals to stay informed about the latest trends and advancements in the field to ensure their systems remain secure in the face of evolving threats. While Kai et al. (2011) suggest TransSQL, which converts web application inquiries into LDAP queries and compares the results with SQL database results. Although it requires no modification of legacy web applications, the approach can be complex and resource-intensive, potentially impacting performance and manageability.

Table 2.6 reveal the schemes and the capabilities of detection and summarize the assessment. The symbol √ refer to the techniques that successfully detect all kinds of malicious attacks within a particular type. The symbol × refer to the techniques that is not capable to detect all attacks of that particular type.

Table 2.6: A comparison between SQLI detection techniques based on the types of attacks.

	SAFELI	SQL-IDS	Swaddler	SQL Prevent	SQLran	SQLIPA	DIWeDa	Tautology Checker	Removing SQL query	SQLCheck	SQL Guard
Tautologies	×	√	×	√	√	√	×	√	√	√	√
Piggy- backed	√	√	×	√	√	×	×	×	√	√	√

	SAFELI	SQL-IDS	Swaddler	SQL Prevent	SQLran	SQLIPA	DIWeDa	Tautology Checker	Removing SQL query	SQLCheck	SQL Guard
Illegal/ Incorrect	√	√	×	√	√	×	×	×	√	√	√
Union	√	√	×	√	√	×	×	×	√	√	√
Stored Procedure	√	√	×	√	×	×	×	×	√	×	×
Inference	√	√	×	√	√	×	√	×	√	√	√
Alternate Encodings	√	√	×	√	√	×	×	×	√	√	√

It was found in past studies that different authors presented their relevant work in SQLI at varying levels of detail; for example, obtaining uniform data from such a broad range of articles was a very difficult operation. The major conclusions from the demonstration of related works reveal that in order to perform a SQLI attack, an attacker must utilize a space, double quotes, or double dashes in his input. The approach for detecting one of the symbols listed above has been discussed.

The previous arguments and demonstrations of detection techniques reveals that if a web developer lacks sufficient security understanding, he or she is likely to write code that contains flaws and affect the security of the database in the remote server. Table 2.6 indicates the comparisons between the available SQLI detection methods found in the literature based on the techniques used to detect malicious SLQ code.

Table 2.7**:** SQLI détection comparaison

Techniques	Developer	Detection outcome	Tautologies	Inference	Logically Incorrect	Alternate encoding	Union query	Backed query	Stored
NSSA	Xu et al., 2017	Prevent/ Detect	X	X	X	X	X	X	X
Deep learning framework	Ding et al., 2021	Detect	X	X			X		X
Tautology CHECKER	Manoj et al., 2014	Detect	X						
PSIAQOP	Al-Khashab et al., 2011	Prevent/ Detect	X	X	X	X	X	X	X
Lexicon /Tokenizes	Hlaing & Khaing, 2020	Prevent/ Detect	X		X	X	X	X	X
SQLiGOT	Kar, 2016	Classification /Detect	X	X	X	X	X	X	X
DE-PRE	Churi & K. Mistry, 2018	Prevent/ Detect	X	X	X	X	X	X	X
CANDID	Bisht et al., 2010	Prevent/ Detect	X	X	X	X	X	X	X
SQL Shield	Mehta et al., 2015	Prevent/ Detect	X	X	X	X	X	X	X
vASQLi	Appelt et al., 2014	Detect	X	X	X	X		X	X
SQLStor	Mamadhan et al., 2012	Prevent							X
Random	Avireddy et al., 2012	Detect	X	X	X	X	X	X	X

Techniques	Developer	Detection outcome	Tautologies	Inference	Logically Incorrect	Alternate encoding	Union query	Backed query	Stored
Dynamic Analyzer	Nadeem et al., 2017	Prevent/ Detect	X	X	X	X	X	X	X
DAS	Das et al., 2019	Detect	X	X	X	X	X	X	X
Modsecurity with ML	Betarte & Martnez, 2018	Prevent/ Detect	X	X	X	X	X	X	X
JoanAudit	Thomé et al., 2018	Prevent/ Detect	X	X	X	X	X	X	X
SecWA	Busch & Wirsing, 2015	Modelization /Detect/Prev ent	X	X	X	X	X	X	X
Ingre	Ingre, 2017	Classificatio n /Detect	X	X	X	X	X	X	X
Firewall-M	Verbruggen and T. Heskes, 2016	Classificatio n /P	X	X	X	X	X	X	X
SEPTIC	Medeiros et al., 2019	Prevent/ Detect						X	X
OBFUSCATI ON	Halder and A. Cortesi, 2010	Detect			X				X
WDIM	Masango et al., 2017	Detect	X	X	X	X	X	X	X
CCSD	Wu et al., 2017	Detect	X	X	X	X	X	X	X

Techniques	Developer	Detection outcome	Tautologies	Inference	Logically Incorrect	Alternate encoding	Union query	Backed query	Stored
MAC based	Diksha & Madhumita, 2015	Detect	X		X	X	X	X	
HIPS	Makiou et al., 2015	Classification /Detect	X	X	X	X	X	X	X
Machine-Learning	Joshi and V. Geetha, 2015	Classification /Detect	X			X	X		
Information Theory	Shahriar & Zulkernine, 2012	Detect	X	X	X	X	X	X	
AutoRand	Perkins et al., 2016	Prevent/ Detect	X				X	X	

2.5.2 Comparative Analysis

Recent studies have increasingly focused on using machine learning and deep learning models to detect SQLi attacks. For instance, Venkatramulu et al. (2024) developed a detection model using word embedding and machine learning algorithms to identify SQLi attack patterns in web and IoT systems. While their study demonstrated promising accuracy in identifying structured malicious queries, it heavily relied on large volumes of labeled training data and computationally expensive model training. This dependency limits its adaptability to unseen or evolving SQLi patterns. In contrast, the current study's proposed DetectCombined technique eliminates such data dependency by introducing a lightweight, real-time detection process composed of filtration, validation, and historical referencing stages. This makes it adaptable to new attack vectors without the need for retraining or data labeling.

Similarly, Abdullah and Abdulazeez (2024) reviewed supervised learning algorithms such as Support Vector Machines (SVM) and Random Forest for SQLi detection. Their findings highlighted significant issues with false positives and the inability of machine learning models to effectively detect zero-day attacks that do not match learned patterns. The DetectCombined approach directly addresses this gap through proactive filtration and encrypted validation mechanisms that analyze user inputs at runtime, thereby detecting both known and unknown SQLi attempts without relying on predefined attack signatures. This adaptive, rule-based system improves real-time accuracy while maintaining simplicity and low computational overhead.

In another study, Arasteh et al. (2024) proposed a hybrid optimization approach that combines the Binary Gray Wolf Optimizer with machine learning algorithms to enhance feature selection and detection precision. Although this method improved accuracy, its complex computational structure made it unsuitable for real-time web environments, especially those handling large user traffic. Conversely, the DetectCombined model is designed for live web applications, using PHP and JavaScript to conduct lightweight detection before queries reach the database. This allows for efficient execution with minimal latency, even under high traffic conditions, thereby addressing a key limitation of optimization-based and deep learning techniques.

Deep learning models have also been explored for SQLi detection. Liu and Dai (2024) implemented a hybrid BERT–LSTM model that captures contextual and semantic relationships in SQL queries, achieving superior detection accuracy. However, their system demands substantial computational resources and suffers from processing delays, making it unsuitable for real-time applications. The DetectCombined technique, in contrast, offers instant detection by embedding filtration and encrypted validation directly within the web application layer, making it more practical for deployment in online systems such as banking portals or e-commerce websites. Similarly, Aburashed et al. (2024) evaluated traditional machine learning classifiers for SQLi detection but noted the lack of adaptive learning in these models. The DetectCombined method bridges this limitation through its “history” component, which logs and references previous injection attempts to identify recurring or modified attack patterns dynamically.

Further research by Bakır (2025) introduced UniEmbed, a machine learning approach that combines multiple feature fusion techniques to detect both XSS and SQLi attacks. While effective across multiple attack types, the model's complexity results in a trade-off between generalization and precision. The DetectCombined model, however, focuses specifically on SQLi detection, ensuring higher precision and efficiency within this scope. Likewise, Shekhar et al. (2025) proposed a hybrid text-mining and generative AI framework for SQLi detection, where generative models simulated attacks to train the detection system. Despite the novelty of this approach, it remains unsuitable for real-time prevention due to high training and inference latency. In contrast, DetectCombined requires no pretraining and operates immediately within web scripts, providing a real-time, adaptive shield against SQLi.

Other recent studies have emphasized rule-based or framework-oriented approaches. Fadlil et al. (2024) utilized the Open Web Application Security Project (OWASP) framework to mitigate SQLi attacks by enforcing secure coding practices and validation routines. While valuable, this method depends heavily on manual developer compliance and periodic updates to remain effective. The DetectCombined approach automates this process by embedding adaptive rule enforcement within the web application code, reducing the reliance on human oversight. Similarly, Alhowiti and Mohamed (2025) proposed a database integrity protection technique designed to secure the database layer against SQL injection attacks. However, because this operates after query submission, it cannot stop malicious inputs before execution. The DetectCombined method fills this gap by intercepting user inputs at the application layer, ensuring that malicious data is filtered before reaching the database engine.

Begum et al. (2025) introduced a modified validation scheme to reinforce web application security against SQLi, focusing primarily on improving the input validation process. While their approach strengthens static validation, it lacks adaptive features capable of learning from new attack behaviors. The DetectCombined model enhances this concept by integrating three sequential modules filtration, validation, and history allowing for continuous learning and adaptability to changing attack signatures. Similarly, Chen et al. (2025) explored deep learning-based content matching for SQLi detection, achieving strong detection accuracy but at the cost of high computational load

and latency. The DetectCombined method overcomes this by employing encrypted validation and history referencing, maintaining a balance between detection accuracy and execution efficiency, making it better suited for real-time use.

Studies such as Ajasa et al. (2025) examined the effects of SQLi attacks on database performance using Docker-based virtualization. Although insightful in assessing system performance degradation due to SQLi, the study did not propose a proactive detection mechanism. The current research contributes a practical solution through the DetectCombined algorithm, which not only strengthens database protection but also minimizes unauthorized access attempts in live environments. Kashyap and Jana (2025) provided a comprehensive survey of SQLi detection and prevention techniques, categorizing them into static, dynamic, and hybrid methods but without implementing or experimentally validating any model. In contrast, the DetectCombined study bridges this theoretical-practical gap by developing, implementing, and empirically evaluating a functional detection model on an online banking portal simulation.

Finally, Kumar et al. (2024) analyzed SQLi vulnerabilities in cloud computing environments, particularly within multi-tenant architectures. Their study emphasized the need for enhanced detection models but did not provide a deployable framework applicable to conventional web systems. The DetectCombined technique contributes in this regard by offering a universally applicable detection strategy for both web and server-side environments. Overall, the majority of the recent studies from 2024 to 2025 rely on advanced computational or data-driven methods, which, while accurate, are not practical for immediate deployment due to their complexity and latency. The DetectCombined method fills this gap by providing a lightweight, adaptive, and real-time detection system that integrates filtration, encryption, and learning capabilities directly into the web application layer.

In summary, the key novelty of this study lies in its ability to combine proactive filtration, encrypted input validation, and adaptive historical learning into one integrated framework that functions in real-time within the web environment. Unlike data-intensive machine learning or deep learning models, DetectCombined requires no external training, ensuring immediate adaptability to novel attacks. Furthermore, it

achieves high detection accuracy with low computational cost, making it scalable and practical for modern web applications. By focusing on execution-level prevention rather than post-detection analysis, this study significantly contributes to strengthening cybersecurity frameworks against one of the most persistent threats in web application security. The summary of recent studies and gaps is indicated in Table 2.8 below:

Table 2.8: Comparison of Recent SQL Injection Detection Studies

Author(s) & Year	Objective of the Study	Key Findings	Identified Gaps / Limitations
Venkatramulu et al. (2024)	To detect SQLi attacks using word embedding and ML algorithms in web and IoT systems.	ML with word embedding effectively detected known SQLi patterns with good accuracy.	Requires large labeled datasets and struggles with unseen (zero-day) SQLi patterns.
Abdullah & Abdulazeez (2024)	To review the performance of supervised ML algorithms (SVM, RF, KNN) for SQLi detection.	ML classifiers achieved high detection accuracy for known attacks.	High false positives; limited generalization; lack of adaptability to new attacks.
Arasteh et al. (2024)	To optimize SQLi detection using Binary Gray Wolf Optimizer combined with ML algorithms.	Feature optimization improved accuracy in static environments.	High computational complexity; unsuitable for real-time implementation.
Liu & Dai (2024)	To develop a hybrid BERT–LSTM model for SQLi detection using deep learning.	Achieved strong semantic understanding and high detection accuracy.	Requires extensive computational resources; latency issues in live web systems.
Aburashed et al. (2024)	To apply various ML algorithms for SQLi detection.	Demonstrated that ML can classify SQLi attempts effectively with sufficient training data.	Static models; no adaptive capability for evolving attack patterns.
Bakır (2025)	To detect both XSS and SQLi using UniEmbed multi-feature fusion with ML.	Provided multi-attack coverage with good accuracy.	Increased model complexity and trade-off between precision and generalization.
Shekhar et al. (2025)	To design a hybrid text-mining and Generative-AI framework for SQLi detection.	Generative AI produced diverse attack simulations, improving training diversity.	Requires high training and inference time; impractical for real-time web defense.
Fadlil et al. (2024)	To mitigate SQLi using the OWASP framework and secure coding standards.	OWASP guidelines enhanced baseline security practices.	Manual and developer-dependent; lacks automated or adaptive detection.
Alhowiti & Mohamed (2025)	To protect database integrity against SQL injection attacks.	Proposed a database-level protection mechanism that prevents data corruption.	Detection occurs after input reaches the database; not preventive at input stage.
Begum et al. (2025)	To reinforce web application security through	Improved static input validation methods reduced	No adaptive learning; static validation limits long-term

Author(s) & Year	Objective of the Study	Key Findings	Identified Gaps / Limitations
	a modified validation scheme.	SQLi success rates.	effectiveness.
Chen et al. (2025)	To detect SQLi using deep learning-based content matching.	Deep learning achieved accurate query analysis and detection.	High computational overhead; limited real-time usability.
Ajasa et al. (2025)	To evaluate the performance effects of SQLi attacks using Docker-based virtualization.	Showed how SQLi affects database speed and performance.	Focused on performance analysis, not proactive detection or prevention.
Kashyap & Jana (2025)	To survey existing SQLi detection and prevention methods.	Provided classification of static, dynamic, and hybrid detection approaches.	Purely theoretical; lacked implementation and empirical testing.
Kumar et al. (2024)	To analyze SQLi vulnerabilities in cloud and web applications.	Identified major attack vectors in multi-tenant and web systems.	No concrete detection mechanism proposed.
Current Study (DetectCombined)	To develop an adaptive SQLi detection technique (DetectCombined) using filtration, validation, and history mechanisms in PHP and JavaScript.	Achieved high detection accuracy, reduced false negatives, and improved real-time prevention through adaptive filtering and encrypted validation.	Slight computational overhead due to encryption and history table management; future improvement through AI integration recommended.
Vastare, A. R., KR, N., & Aiman, U. (2025)	To compare traditional and AI-based techniques for detecting and preventing SQL injection attacks.	AI-based approaches demonstrated higher detection accuracy and adaptability than rule-based systems.	Limited evaluation on real-world dynamic web applications and lack of benchmarking against emerging attack patterns.
Rosca, C. M., Stancu, A., & Popescu, C. (2025)	To develop and evaluate machine learning models for detecting SQL injection patterns.	ML models such as Random Forest and SVM achieved superior detection rates and reduced false positives.	Did not address scalability and model retraining issues for continuously evolving SQL injection variants.
AlZaidi, J., & Janbi, J. A. (2025)	To propose a Probabilistic Neural Network (PNN)-based model for detecting SQL injection attacks.	PNN effectively classified malicious inputs and outperformed conventional classifiers.	The study lacked comparative performance analysis with deep learning architectures and real-time validation.
Mehmood, A., Hasan, M. Z., & Hussain, M. Z. (2025)	To present comprehensive detection and defense strategies to secure	Introduced a hybrid detection model combining static and dynamic analysis for	The model's implementation complexity and high computational cost were

Author(s) & Year	Objective of the Study	Key Findings	Identified Gaps / Limitations
	databases from SQL injection.	enhanced protection.	not optimized for low-resource systems.
Hossain, M. M., Hasnat, M. A., & Islam, M. S. (2025)	To enhance SQL injection prevention through innovative query comparison and encryption algorithms.	The algorithm improved query validation and data confidentiality.	Lack of empirical validation on large-scale, high-traffic web systems limits generalizability.
Qin, Q. et al. (2025)	To design a Bi-LSTM-based SQL injection detection algorithm with integrated feature selection.	The Bi-LSTM model demonstrated high accuracy and robustness against obfuscated attacks.	High training time and dependency on large labeled datasets remain major challenges.
Lo, R. T., Hwang, W. J., & Tai, T. M. (2025)	To develop a lightweight SQL injection detection model using Multi-Head Self-Attention (MHSA).	MHSA-based model achieved faster detection with minimal computational resources.	Evaluation focused on synthetic datasets; real-world web traffic validation is needed.
AL SALMAWI, H. M. A. (2025)	To critically evaluate the effectiveness of current SQL injection security measures in web applications.	Revealed that most commercial web apps rely on outdated sanitization and lack robust intrusion detection.	Did not propose an alternative or improved detection framework for future use.
Hatture, S. M., Ari, M., Biju, A., & Gayathri, N. V. (2025)	To design a "Secure Scan" vulnerability scanner for identifying SQL injection weaknesses.	Secure Scan successfully detected vulnerabilities in PHP-based systems with high precision.	Needs enhancement for multi-language web environments and real-time automated patching.
Prosper, J. (2025)	To integrate AI and encryption methods for SQL injection detection and broader cybersecurity applications in smart cities.	AI-encryption integration improved system-wide resilience across web and IoT layers.	Lacked domain-specific evaluation for web-only SQL injection threats.
Anny, D. (2025)	To propose a multimodal AI and blockchain-based security framework addressing SQL	Combined AI and blockchain improved data traceability and threat detection synergy.	The broad scope reduced SQL injection–specific technical depth and testing coverage.

Author(s) & Year	Objective of the Study	Key Findings	Identified Gaps / Limitations
	injection, deepfake, and privacy threats.		
Kumar, L., & Srivastava, P. (2025)	To optimize SQL injection prevention using a multi-layered defense approach.	Multi-layered defense integrating firewalls, input validation, and ML reduced risk exposure.	Requires further testing under zero-day and adaptive attack conditions to confirm resilience.

2.5.3 Real-world examples of successful implementations of SQL detection technique in protecting web applications.

One notable example is the case of the social media platform Facebook, which faced numerous SQL injection attacks in its early days. Through implementing strict input validation and parameterized queries, Facebook was able to effectively thwart these attacks and protect its users' data. Another example is the e-commerce giant Amazon, which also experienced SQL injection vulnerabilities in the past. Through continuous monitoring and regular security audits, Amazon was able to identify and patch these vulnerabilities before any significant damage could occur. These real-world examples highlight the importance of proactive measures in safeguarding web applications from SQL injection threats. (Qu et al., 2024). By learning from these experiences, other companies can also strengthen their defences against potential SQL injection vulnerabilities. By prioritizing security measures such as regular audits, implementing best practices, and staying updated on the latest threats, organizations can minimize the risk of falling victim to such attacks. It is crucial for businesses to invest in robust security protocols to ensure the safety and confidentiality of customer data, ultimately earning trust and loyalty from their user base (Sinha & Verma, 2021).

Implementing strong encryption techniques and access controls can also help mitigate the risks associated with SQL injection attacks. Additionally, regularly monitoring and updating security protocols can help organizations stay ahead of evolving threats and maintain a secure online environment for their customers. By taking a proactive approach to security, companies can demonstrate their commitment

to protecting sensitive information and safeguarding against potential breaches. This not only helps to protect the reputation of the business, but also ensures compliance with industry regulations and standards (Lu et al., 2023). In other sense, investing in comprehensive security measures is essential in today's digital landscape to prevent costly data breaches and maintain customer trust. Implementing regular security audits and training sessions for employees can also greatly enhance a company's overall security posture. By staying informed about the latest cybersecurity trends and threats, organizations can better anticipate and prevent potential attacks. Additionally, establishing clear incident response plans and regularly testing them can help mitigate the impact of any security incidents that do occur. Ultimately, prioritizing cybersecurity measures is crucial for maintaining a strong and trustworthy online presence in today's increasingly interconnected world (Crişan et al., 2020).

Successful implementations of SQL injection protection techniques in real-world applications highlight the critical importance of using best practices to secure web applications. One notable example is the case of CardSystems Solutions, a third-party payment processor for major credit card companies like MasterCard and VISA. In 2005, an attacker exploited an SQL injection vulnerability, leading to the theft of 40 million credit card numbers. This breach underscored the need for robust SQL injection defences, such as encrypting sensitive data and using parameterized queries (Qualys, 2024).

Another significant example is the protection measures implemented by Sony. In 2011, Sony's PlayStation Network was compromised due to an SQL injection attack, resulting in the theft of personal data from approximately 77 million accounts. Following this breach, Sony strengthened its defences by adopting prepared statements and parameterized queries, reducing the risk of similar attacks in the future (DZone, 2023). Tesla has also been a target of SQL injection attacks. In 2014, Tesla's website had a vulnerability that could have been exploited to execute arbitrary SQL commands. The company mitigated this risk by implementing input validation and parameterized queries, thereby preventing unauthorized access and data manipulation (DZone, 2023).

Cisco's deployment of security measures against SQL injection is another pertinent example. Cisco's approach includes a multi-layered defence strategy involving

input validation, the use of prepared statements, and regular security assessments to identify and mitigate vulnerabilities. These measures have proven effective in maintaining the integrity and confidentiality of their systems (HackerOne, 2024). In addition to these cases, the protection of web applications from SQL injection can be further enhanced by adopting practices such as escaping special characters, using whitelists to validate input, and employing tools like Web Application Firewalls (WAFs). These practices are essential in creating a robust security posture against SQL injection attacks, as evidenced by various industry implementations (HackTheBox, 2024).

Recent real-world implementations highlight the efficacy of various techniques in protecting web applications from SQL injection vulnerabilities. For example, one notable method involves the use of prepared statements and parameterized queries, which prevent attackers from injecting malicious SQL code into queries. This approach has proven effective because it forces developers to define SQL code first and then pass each parameter to the query later, thus separating code from data and rendering the input harmless if it contains SQL commands (Brightsec, 2023).

Another successful technique is the adoption of Web Application Firewalls (WAFs). WAFs act as a barrier between web applications and the internet, filtering out malicious traffic. This method has been implemented effectively by many enterprises, providing an additional layer of security that monitors and analyzes incoming HTTP requests to block SQL injection attempts. Companies like Bitdefender have included WAFs in their enterprise antivirus solutions, thereby enhancing their overall security posture (Safety Detectives, 2024). Furthermore, regular security updates and patches are crucial in preventing SQL injection attacks. For instance, WordPress, which powers a significant portion of the web, constantly patches vulnerabilities with each new version. This proactive approach minimizes the risk of SQL injection by ensuring that known vulnerabilities are fixed promptly (Safety Detectives, 2024).

Recent reports have also emphasized the importance of input validation and sanitation. By strictly validating and sanitizing user inputs, web applications can prevent attackers from injecting harmful SQL code. This method was effectively used to thwart a potential SQL injection attack in a case involving a popular e-commerce

platform. The platform's developers ensured that input fields accepted only expected data types and rejected any inputs containing potentially harmful characters (Brightsec, 2023; Pentest-Tools.com, 2024). These examples underscore the significance of a multi-faceted approach to SQL injection prevention, combining technical safeguards like prepared statements and WAFs with organizational practices such as regular updates and robust input validation protocols. By adopting such comprehensive strategies, organizations can significantly reduce the risk of SQL injection attacks and enhance the security of their web applications (PortSwigger, 2024).

2.6 Limitations and challenges of SQLI approach threats targeting databases.

One of the main limitations of using this new approach in defending against cyber threats targeting databases is the potential for false positives. This means that the system may incorrectly flag legitimate activity as suspicious, leading to unnecessary investigations and potentially disrupting normal operations. Additionally, the complexity and sophistication of modern cyber threats may require constant updates and adjustments to the defence mechanisms in order to effectively mitigate risks. This can be challenging for organizations with limited resources or expertise in cybersecurity, as they may struggle to keep up with the evolving threat landscape. Furthermore, the implementation of this new approach may require significant investment in training and infrastructure, which could be a barrier for some organizations. Despite these challenges, however, the potential benefits of using this new approach to defend against cyber threats targeting databases are significant and can greatly enhance an organization's security posture. By investing in proper training and infrastructure, organizations can better protect their databases and sensitive information from cyber threats. Utilizing this new approach can also help organizations stay ahead of cybercriminals and minimize the potential impact of a data breach. Ultimately, the investment in cybersecurity measures will pay off in the long run by safeguarding the organization's reputation, financial stability, and overall success.

In addition to investing in training and infrastructure, organizations should also prioritize regularly updating their security protocols and staying informed about the latest cyber threats. Implementing multi-factor authentication, encryption, and regular

security audits can further strengthen a database's defences against potential attacks. By taking a proactive approach to cybersecurity, organizations can better mitigate risks and protect their valuable assets. Furthermore, fostering a culture of cybersecurity awareness among employees can help prevent human error and reduce the likelihood of a successful cyber-attack. Overall, a comprehensive and holistic approach to defending against cyber threats is essential for maintaining a secure and resilient organization in today's digital landscape. Implementing multi-factor authentication, utilizing strong encryption protocols, and regularly monitoring and assessing security measures are crucial steps in safeguarding sensitive data. By investing in cutting-edge security technologies and instilling a sense of responsibility for cybersecurity throughout the organization, companies can significantly reduce the likelihood of falling victim to cyber threats. Ultimately, creating a robust cybersecurity framework is imperative for staying ahead of evolving threats and ensuring the long-term success of the business.

Adopting new approaches to defend against cyber threats targeting databases presents several challenges and limitations. One of the primary challenges is the dynamic and evolving nature of cyber threats, which makes it difficult for new methods to remain effective over time. Cyber attackers constantly develop new techniques and strategies, requiring continuous updates and improvements to defence mechanisms. This constant need for adaptation can strain resources and require significant investment in research and development (Ebrahimi, 2021).

Another significant limitation is the reliance on artificial intelligence (AI) and machine learning (ML) techniques in modern cybersecurity approaches. While AI and ML can enhance threat detection and response capabilities, they are not infallible. These systems are vulnerable to adversarial attacks where attackers manipulate input data to deceive AI models. This vulnerability can lead to false positives or negatives, potentially allowing threats to go undetected or causing unnecessary alerts and responses (Ebrahimi, 2021). Moreover, integrating new cybersecurity measures often requires substantial changes to existing IT infrastructure, which can be costly and time-consuming. Organizations may face operational disruptions during the implementation phase, and the complexity of new systems may necessitate specialized training for

cybersecurity personnel. This transition period can be a window of opportunity for attackers to exploit vulnerabilities (SpringerLink, 2018).

Finally, the effectiveness of new cybersecurity approaches heavily depends on the quality and comprehensiveness of the threat intelligence they rely on. Incomplete or inaccurate threat intelligence can undermine the entire defence strategy, leading to gaps in protection. The collection and analysis of high-quality threat intelligence require robust frameworks and collaboration across different sectors, which can be challenging to establish and maintain (Nature, 2023). In other words, while new approaches to defending against cyber threats targeting databases hold promise, they come with significant challenges related to the evolving threat landscape, vulnerabilities of AI and ML systems, integration complexities, and the need for high-quality threat intelligence (Ebrahimi, 2021; SpringerLink, 2018; Nature, 2023). Adopting new approaches to defend against cyber threats targeting databases involves addressing several critical challenges and limitations that can affect their efficacy and implementation. These challenges include technological, operational, and strategic aspects.

Technological Challenges

1. **AI and Machine Learning Vulnerabilities**: While AI and ML have revolutionized cybersecurity by enabling rapid threat detection and automated responses, they are susceptible to adversarial attacks. Adversaries can manipulate data inputs to mislead AI models, causing them to fail in detecting threats or generating false positives. This issue is critical because it undermines the reliability of AI-based defence mechanisms (Ebrahimi, 2021). Furthermore, AI models require vast amounts of high-quality data to function effectively, and any deficiency in the data quality can significantly degrade their performance (SpringerLink, 2018).
2. **Complexity and Integration**: Implementing new cybersecurity approaches often necessitates significant changes to existing IT infrastructure. This can be particularly challenging for organizations with legacy systems, as integrating new technologies with old systems can lead to compatibility issues and operational disruptions. Additionally, the complexity of these systems may

require specialized training for staff, which can be resource-intensive and time-consuming (Nature, 2023).

Operational Challenges

1. **Resource Constraints**: Continuous updates and improvements are necessary to keep pace with the evolving threat landscape. This ongoing need can strain organizational resources, both in terms of finances and personnel. Smaller organizations, in particular, may struggle to allocate sufficient resources to maintain robust cybersecurity defences (SpringerLink, 2018).
2. **Incident Response and Recovery**: New cybersecurity measures must not only detect and prevent threats but also support effective incident response and recovery. In many cases, organizations may find that their incident response plans are not well-integrated with new technologies, leading to delays and inefficiencies in responding to cyber incidents (Ebrahimi, 2021).

Strategic Challenges

1. **Quality of Threat Intelligence**: The success of new cybersecurity approaches heavily depends on the quality and comprehensiveness of threat intelligence. Incomplete or inaccurate threat intelligence can create gaps in protection, allowing threats to bypass defences. Establishing robust frameworks for threat intelligence collection and analysis requires extensive collaboration across various sectors, which can be challenging to achieve and sustain (Nature, 2023).
2. **Regulatory and Compliance Issues**: Organizations must also navigate a complex landscape of regulatory and compliance requirements related to cybersecurity. Ensuring that new defence mechanisms comply with these regulations can be challenging, particularly when regulations vary significantly across different regions and industries. Non-compliance can result in severe legal and financial penalties, further complicating the implementation of new cybersecurity measures (Ebrahimi, 2021).

Strategic Implications

1. **Evolving Threat Landscape**: The cyber threat landscape is constantly evolving, with attackers developing new techniques and strategies at a rapid pace. This dynamic environment necessitates continuous monitoring and

adaptation of defence mechanisms. Organizations must be agile and proactive in their approach to cybersecurity, regularly updating their defences to address emerging threats (SpringerLink, 2018).

2. **Collaborative Defence Efforts**: Addressing cyber threats effectively requires a collaborative approach, involving partnerships between organizations, governments, and cybersecurity firms. Sharing threat intelligence and best practices can enhance the overall cybersecurity posture of all stakeholders involved. However, achieving effective collaboration can be challenging due to issues related to trust, data privacy, and information sharing agreements (Nature, 2023).

In summary, while new approaches to defending against cyber threats targeting databases offer significant potential benefits, they also come with critical challenges related to technological vulnerabilities, resource constraints, integration complexities, and the need for high-quality threat intelligence and regulatory compliance. Addressing these challenges requires a comprehensive and proactive approach, involving continuous monitoring, collaboration, and investment in research and development (Ebrahimi, 2021; SpringerLink, 2018; Nature, 2023).

2.7 The impact of malicious SQL injection attacks on businesses and organizations, including financial losses and damage to reputation.

These attacks can result in stolen sensitive information, disrupted operations, and compromised customer trust. In addition, the costs associated with resolving the aftermath of a SQL injection attack can be significant, including expenses for forensic investigations, regulatory fines, and legal fees. As a result, businesses must prioritize implementing robust security measures and regularly updating their systems to protect against these types of cyber threats. By staying informed about the latest security threats and vulnerabilities, businesses can take proactive steps to safeguard their data and systems. Investing in employee training on cybersecurity best practices and enforcing strict access controls can also help minimize the risk of falling victim to SQL injection attacks. Ultimately, by taking a proactive approach to cybersecurity, businesses can protect themselves from the potentially devastating consequences of a successful attack (Abdullayev & Chauhan, 2023).

In addition to investing in employee training and enforcing access controls, businesses should also consider implementing regular security audits and penetration testing to identify and address any potential vulnerabilities in their systems. By conducting regular audits, businesses can stay ahead of emerging threats and ensure that their security measures are up to date. Furthermore, establishing a response plan for handling security incidents can help minimize the impact of a successful attack and facilitate a swift recovery process. By taking a comprehensive and proactive approach to cybersecurity, businesses can better protect themselves from the ever-evolving landscape of cyber threats (Jacob & Pirnau, 2020).

SQL injection (SQLi) attacks have become one of the most prevalent and damaging forms of cyberattacks faced by businesses and organizations today. These attacks exploit vulnerabilities in an application's software by manipulating SQL queries, allowing attackers to gain unauthorized access to sensitive data. The impact on businesses can be catastrophic, leading to significant financial losses and severe damage to their reputation. According to recent studies, the average cost of a data breach caused by an SQL injection attack can range from thousands to millions of dollars, depending on the size and nature of the business affected (IBM Security, 2022). Furthermore, the aftermath of an SQL injection attack can also involve legal repercussions, as businesses may face lawsuits from customers whose personal information was compromised. In addition to financial losses, businesses must invest in cybersecurity measures to prevent future attacks and rebuild trust with their customers. It is crucial for organizations to prioritize cybersecurity and regularly update their software to mitigate the risk of SQL injection attacks and protect sensitive data from falling into the wrong hands.

Financial losses stem not only from the immediate fallout of the attack but also from the long-term costs associated with remediation, legal actions, and potential fines from regulatory bodies. Organizations may also face increased costs for cybersecurity insurance premiums following an attack. Moreover, businesses often experience a decline in stock prices post-breach, reflecting the loss of investor confidence. For instance, the infamous SQL injection attack on Heartland Payment Systems in 2009 resulted in a staggering $12.6 million in legal settlements and remediation costs (Verizon, 2023). As a result, organizations must not only invest in strengthening their

cybersecurity defences to prevent future attacks, but also prepare financially for the potential aftermath of a breach. The financial implications of a cyberattack can have long-lasting effects on a company's bottom line and reputation, making it crucial for businesses to prioritize cybersecurity measures and response strategies. Ultimately, the true cost of a cyberattack goes far beyond the immediate damages and can impact the overall financial health and sustainability of an organization.

Beyond financial repercussions, SQL injection attacks significantly harm the reputation of the affected organizations. Customers and clients lose trust in a company's ability to protect their personal and financial information, which can lead to a decline in customer loyalty and a drop in sales. A survey conducted by Ponemon Institute in 2023 revealed that 60% of consumers indicated they would likely discontinue their relationship with a company if their data were compromised in a breach. This loss of trust can be particularly devastating for businesses in highly competitive markets, where customer retention is crucial for long-term success (Ponemon Institute, 2023). In addition to losing customers, businesses that experience data breaches may also face costly legal fees and fines, as well as damage to their brand reputation that can take years to repair. In some cases, data breaches can even lead to the closure of a business altogether. Therefore, it is essential for organizations to prioritize cybersecurity measures and invest in robust data protection strategies to safeguard against potential breaches and mitigate the damaging effects they can have on their operations. By taking proactive steps to secure their systems and data, businesses can protect their customers, their bottom line, and their future success in the marketplace.

The reputational damage is further exacerbated by the negative publicity surrounding such incidents. News of a data breach spreads quickly, and companies often find themselves under intense scrutiny from both the media and the public. This negative attention can lead to a tarnished brand image and long-term damage that may take years to repair. In many cases, companies must invest heavily in marketing and public relations efforts to restore their reputation and regain customer trust (Symantec, 2023). Implementing robust cybersecurity measures is essential for businesses to safeguard their data and prevent potential breaches. By taking proactive steps to protect sensitive information, companies can avoid the costly consequences of a data breach

and maintain the trust of their customers. Investing in cybersecurity not only protects the company's reputation but also demonstrates a commitment to safeguarding customer data, ultimately contributing to long-term success in the marketplace. In today's digital age, prioritizing cybersecurity is not just a best practice - it's a necessity for businesses looking to thrive in an increasingly interconnected world.

To mitigate these risks, businesses must implement robust cybersecurity measures, including regular security audits, employee training, and the adoption of advanced technologies such as machine learning to detect and prevent SQL injection attacks. By proactively addressing these vulnerabilities, organizations can better protect themselves from the financial and reputational damage associated with SQL injection attacks (Check Point Software Technologies, 2023). Furthermore, establishing a strong incident response plan is crucial for businesses to quickly and effectively address any security breaches that may occur. By having a well-defined plan in place, organizations can minimize the impact of SQL injection attacks and swiftly contain any potential damage. In addition, staying informed about the latest cybersecurity threats and trends can help businesses stay one step ahead of cybercriminals and ensure their systems remain secure. Overall, investing in comprehensive cybersecurity measures is essential for safeguarding sensitive data and maintaining the trust of customers and stakeholders.

2.8 Strategies for preventing SQL injection attacks, such as input validation, parameterized queries, and using web application firewalls.

One of the most effective strategies for preventing SQL injection attacks is input validation. By validating all user input before it is passed to the database, developers can ensure that malicious code is not able to be executed. This can be done by checking for specific patterns or formats in the input data, such as ensuring that only alphanumeric characters are allowed in a username field. Parameterized queries are another important defence against SQL injection attacks. By using parameterized queries, developers can separate the SQL code from the user input, preventing attackers from injecting their own malicious code into the query. Additionally, using web application firewalls can help to detect and block SQL injection attacks before they

reach the database. These firewalls can analyze incoming requests and block any that appear to be attempting SQL injection. By implementing a combination of these strategies, developers can greatly reduce the risk of SQL injection attacks on their web applications. Regularly updating and patching software and libraries can also help prevent SQL injection attacks, as vulnerabilities in outdated software can be exploited by attackers. It is important for developers to stay informed about the latest security threats and best practices in order to effectively protect their web applications. In addition, educating users about safe browsing habits and the importance of strong, unique passwords can also help prevent unauthorized access to sensitive data. By taking a proactive approach to security, developers can greatly reduce the likelihood of falling victim to SQL injection attacks.

SQL injection attacks exploit vulnerabilities in web applications by inserting malicious SQL code into input fields. Effective strategies for preventing these attacks include input validation, parameterized queries, and the use of web application firewalls (WAFs).

Input Validation: Input validation is the first line of defence against SQL injection. It involves verifying that user inputs conform to expected formats and rejecting inputs that could be malicious. This strategy ensures that only properly formatted data is processed by the application. Implementing input validation can mitigate the risk of SQL injection by preventing attackers from submitting harmful SQL commands through input fields (Halfond et al., 2006). Parameterized queries: Parameterized queries are another effective way to prevent SQL injection attacks. By using parameterized queries, developers can separate SQL code from user input, reducing the likelihood of malicious commands being injected into the database. This technique binds user input to parameters in the SQL query, preventing attackers from manipulating the query structure. Additionally, web application firewalls (WAFs) can provide an added layer of security by monitoring and filtering incoming traffic to block known SQL injection attacks. By implementing these strategies, organizations can better protect their web applications from SQL injection vulnerabilities and potential data breaches. Moreover, using a whitelist approach, where only known good inputs are accepted, can further strengthen security. Input validation, however, should not be the

sole protective measure as it might not cover all possible attack vectors (OWASP, 2022). Therefore, it is essential for organizations to regularly update their WAFs and implement other security measures, such as regular security audits and penetration testing, to ensure that their web applications are secure against evolving threats. In addition to a whitelist approach, organizations can also consider implementing strong authentication mechanisms and encryption protocols to further enhance their overall security posture. By taking a proactive and multi-layered approach to security, organizations can reduce the risk of SQL injection attacks and safeguard sensitive data from potential breaches.

Parameterized Queries: Parameterized queries, also known as prepared statements, involve using placeholders in SQL queries instead of directly incorporating user inputs. This technique prevents attackers from altering the structure of the SQL query. When the SQL engine processes a parameterized query, it treats user inputs strictly as data, eliminating the possibility of executing malicious code. Parameterized queries are widely recognized as one of the most effective methods to prevent SQL injection (Lindstrom et al., 2019). By implementing parameterized queries, organizations can significantly reduce the risk of SQL injection attacks and enhance the security of their databases. This practice not only protects sensitive information but also maintains the integrity and reliability of the database system. In addition, regularly updating and patching software, employing encryption, and conducting regular security audits can further enhance data protection measures against potential breaches. For instance, in languages like Java, using the Prepared Statement class ensures that SQL commands are separated from data inputs, significantly reducing vulnerability (Shahriar & Zulkernine, 2012).

Web Application Firewalls (WAFs): WAFs provide an additional layer of security by monitoring and filtering HTTP traffic to and from a web application. These firewalls can detect and block malicious inputs before they reach the application. WAFs use a set of predefined rules to identify common attack patterns and can be customized to protect against specific threats. By analyzing traffic and blocking suspicious activities, WAFs help protect applications from SQL injection attacks (Stuttard & Pinto, 2011). In addition to preventing SQL injection attacks, WAFs can also protect against

cross-site scripting (XSS), cross-site request forgery (CSRF), and other common web application vulnerabilities. They act as a barrier between the application and potential attackers, continuously monitoring and filtering traffic to ensure only legitimate requests are processed. By implementing a WAF, organizations can enhance their overall security posture and reduce the risk of data breaches and unauthorized access to sensitive information. However, while WAFs are effective, they should complement, not replace, other security measures like input validation and parameterized queries to ensure comprehensive protection (Kaur & Kaur, 2019).

Combining Strategies: A robust defence against SQL injection attacks involves combining multiple strategies. Input validation, parameterized queries, and WAFs each address different aspects of potential vulnerabilities and together form a comprehensive security framework. Relying solely on one technique could leave gaps that knowledgeable attackers could exploit. Therefore, adopting a multi-layered security approach is essential for reducing the risk of SQL injection attacks and enhancing overall web application security (Su, 2007). In addition to input validation, parameterized queries, and WAFs, regularly updating software and patches, implementing strong encryption, and conducting routine security audits are also crucial components of a multi-layered security strategy. By continuously monitoring and adapting to evolving threats, organizations can stay one step ahead of potential attackers and better protect their web applications from SQL injection attacks. It is important to prioritize security measures and invest in comprehensive protection to safeguard sensitive data and maintain the trust of users.

2.9 Chapter summary

In this chapter, a discussion on various types of SQLI attacks, vulnerabilities, as well as detection and prevention techniques have been presented. The discussion in this chapter demonstrates almost all problems and issues related to SQLI by providing a broad overview and taxonomy of the most popular and latest methods for detecting and preventing these malicious attacks. A sorting of SQLI attack has been extensively

discussed and compared with the technique that have been developed by researcher to prevent them. Each solution's primary characteristics and flaws were also noted.

It has been found that web developers should understand how SQLI works before attempting to beat it. Because of a lack of training and development expertise, web developers may make mistakes when utilizing SQL, particularly inadequate input validation, making the web application highly vulnerable. As a result, it is recommended that during the development process of a dynamic web application, web developers construct dynamic queries by concatenating the SQL statement with variables in an effective approach to prevent SQLI. These variables come from outside the program and allow the software to make dynamic queries based on user conditions at runtime. Many studies in this field discovered that SQLI attacks occur because web developers did not set sufficient restrictions on the input parameters (user name/ password) in the web application, allowing the attacker to insert and run his/her desired query. As a result, the study's proposed remedy is to limit inputs for accessing the database of e-commerce websites. Finally, it is critical to understand that in order to exploit a SQLI vulnerability, the attacker must have simple access to a parameter that the web application sends to the database. The attacker will cause the web application to send the malicious query to the database server and execute it by attaching the malicious SQL commands to the parameter. The query will be vulnerable to SQLI attack in the absence of input refinement. These variables may be informed by a POST or GET request, HTTP Headers or HTTP Cookies.

CHAPTER THREE
RESEARCH METHODOLOGY

3.1 Introduction

This chapter describe the research methods and stages that used for preventing SQL injection on target database. In addition to that this chapter highlights the strength of this method and step by step implementation of this method on the client page. The following sections described the method that will be used to prevent SQL injection.

3.2 Research Design

This study employs a Design Science Research (DSR) methodology, which is particularly suitable for developing and evaluating innovative artifacts aimed at solving practical and technical problems. DSR emphasizes the creation and iterative refinement of solutions that are both relevant in application and rigorous in design. In the context of this study, the central artifact is the DetectCombined technique
an applied, multi-layered SQL injection prevention method that integrates parameterized queries, input filtering, and encryption.

The selection of a design science approach is grounded in the nature of the research objective, which is not merely to understand a phenomenon but to create a working solution to a persistent problem in web application security. SQL injection remains one of the most critical threats identified by organizations such as OWASP, and the existing countermeasures often fall short in dynamic or large-scale environments. The research thus follows the DSR cycle of problem identification, artifact design, demonstration, and evaluation. DetectCombined is developed through iterative design stages, each informed by existing literature and the limitations of current techniques.

The design process begins with identifying weaknesses in conventional methods such as syntax-based filtering, inconsistent use of parameterized queries, and standalone machine learning models. These insights guide the structured development of DetectCombined, which includes multiple security stages registration, login, search,

and encryption. The proposed method is implemented using PHP and JavaScript, and integrates AES encryption to protect fields vulnerable to injection. Each stage of DetectCombined is designed to counter a specific attack vector, ensuring a holistic and layered defence approach.

The artifact is demonstrated by simulating a user interaction environment where malicious SQL inputs are introduced during various phases of input processing. This enables the practical evaluation of the artifact's performance in filtering, blocking, and encrypting potentially harmful queries. While quantitative benchmarking may not be within the scope of this study, the effectiveness of the method is validated through controlled examples that highlight its behavior against typical SQL injection payloads.

In summary, the use of the Design Science Research methodology allows this study to move beyond theoretical exploration and deliver a functional, reproducible, and extensible security framework. DetectCombined is not presented as a theoretical innovation but rather as an applied solution grounded in existing knowledge, tested in implementation, and structured for adaptation in real-world web application environments.

Figure 3.1 illustrates the architectural design of the proposed DetectCombined system, based on a Design Science Research methodology. The architecture is composed of three main components: Client-side (JavaScript), Server-side (PHP), and the Database with integrated Logging.

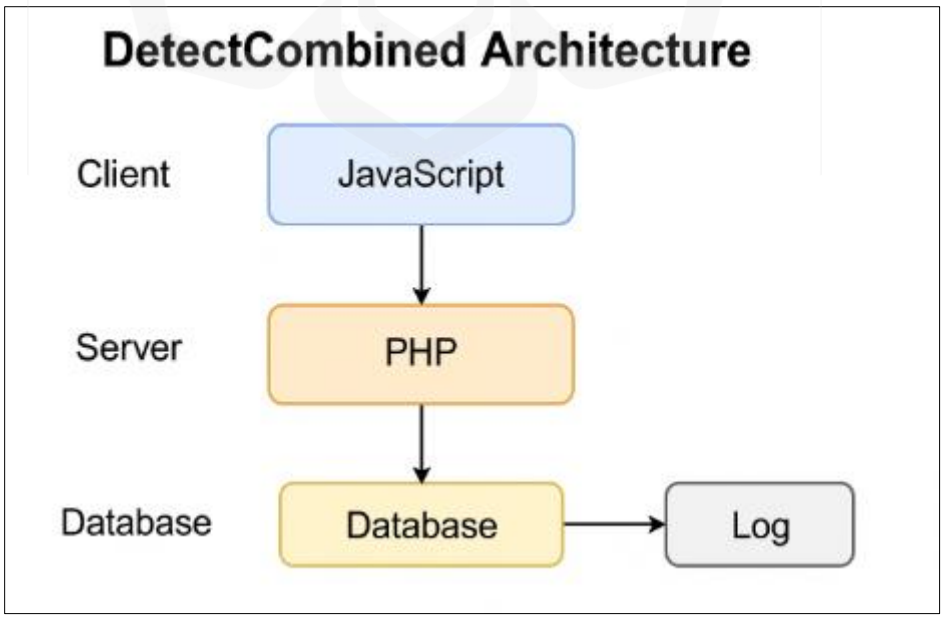


Figure 3.1: DetectCombined architecture

On the client-side, JavaScript handles preliminary input validation by filtering suspicious characters and known SQL injection patterns before data is submitted. This step improves efficiency by reducing unnecessary requests to the server and acts as a first layer of defence. Once the request reaches the server-side, PHP scripts perform further validation and implement parameterized queries to securely interact with the database. This prevents malicious input from being executed as part of an SQL command. During this stage, AES-based encryption is also applied to sensitive fields such as user credentials.

The validated and encrypted queries are then executed against the database, which stores sensitive data in encrypted form. Additionally, the system logs all incoming SQL requests, especially those flagged as suspicious, into a dedicated logging component. This supports real-time analysis and future forensic investigation, reinforcing the security posture of the system. The layered nature of the architecture ensures that SQL injection prevention is not reliant on a single method but instead combines client-side filtering, secure server-side query handling, and encrypted data storage. This structure reflects the Design Science Research approach, emphasizing the development and demonstration of a practical, working artifact to address real-world SQL injection threats.

3.3 Method of the study

Many detection and prevention schemes have been developed proposed to address SQL Injections. These methods typically follow different techniques and many of them were not consistent with large number of users' request to login to a web page. Despite many of them can be implemented in various system and languages and ensure good degree of protection at various levels (server, network, applications). For example, some methods are applicable to certain types of malicious injection attacks such as SQL Prevent, XPath Injection, SQLCheck, and Cross-Site-Scripting (XSS) which were described in previous chapter. The review of literature on SQLIA in previous chapter

shows limited flexible methods that can defeat these attacks. This is due to the power of SQL and the flexibility that it gives to the user a tolerance to enter special characters that can damage the target database. In other words, it seems a method to prevent SQLIA should be more efficient and more flexible to fight against SQL injection.

Below is a detailed pseudocode and a flowchart-style description that illustrate the core algorithm behind the DetectCombined technique. These demonstrate how user inputs are filtered, validated, encrypted, and compared to historical attack patterns for enhanced SQL injection prevention.

```
Function HandleUserInput(inputData):
    // Step 1: Client-Side Filtering
    If containsBlacklistedCharacters(inputData):
        BlockRequest("Potential injection detected (client-side)")
        Exit

    // Step 2: Server-Side Validation
    Sanitize(inputData)  // Escapes and normalizes special characters

    // Step 3: Parameterized Query Preparation
    query = "SELECT * FROM users WHERE username = ? AND password = ?"
    ExecutePreparedStatement(query, inputData.username, inputData.password)

    // Step 4: Check for Known Injection Patterns
    If MatchWithHistoricalPatterns(inputData):
        LogSuspiciousAttempt(inputData, "Pattern matched with historical attack")
        AlertAdmin()

    // Step 5: Encrypt Fields Before Insertion or Querying
    encryptedUsername = AESEncrypt(inputData.username, GenerateKey(inputData))
```

encryptedPassword = AESEncrypt(inputData.password, GenerateKey(inputData))

// Step 6: Query Database with Encrypted Inputs

query = "SELECT * FROM users WHERE encrypted_username = ? AND encrypted_password = ?"

result = ExecutePreparedStatement(query, encryptedUsername, encryptedPassword)

If result is Empty:
  Return "Login Failed"
Else:
  Return "Access Granted"

The solution presented in this chapter can increase SQL injection protection while also allowing for huge amounts of user data to be entered. The method proposed in this work is known as DetectCombined composed. Filtering queries with parameters: Using parameterized queries to protect against SQL injection is a safe method. After the login stage, the developer must utilize JavaScript to validate the input field. Some SQL query placeholders are kept in this manner, particularly for SQL statement variables. The SQL engine will first interpret and compile the query without variables, then save the result. Following this, the approach adds variables and recompiles them. In this situation, the attacker may attempt to enter a malicious query straight into the variable, utilizing the variable as a transport means to convey the malicious code to the SQL query that uses this variable; the SQL engine will then treat the value of this variable as an ordinary string.

The process consists of four parts, beginning with entering data into the web page's input area. The user then begins to enter his information into the username and password input sections before beginning to login. Following this stage, the system created in this study will start validating input by deleting any questionable characters

such as ' OR '1'='1. A smart hacker might get access to all the user names and passwords in a database by simply inserting 105 or 1=1 into the input box, or by simply inserting “or ““=“ into the user name or password text box. The result as follow:

SELECT * FROM Users WHERE Name =”” or “”=”” AND Pass =”” or “”=””

The result SQL is valid. It will return all rows from the table Users, since WHERE ““=““ is always true. In this stage, the filtration technique is based on a “blacklist” of words or characters to search for in SQL input, to prevent SQL injection attacks. The following figure shows all stages of the method of the study.

The following stages shows the stages of preventing SQL injection using ***DetectCombined*** method

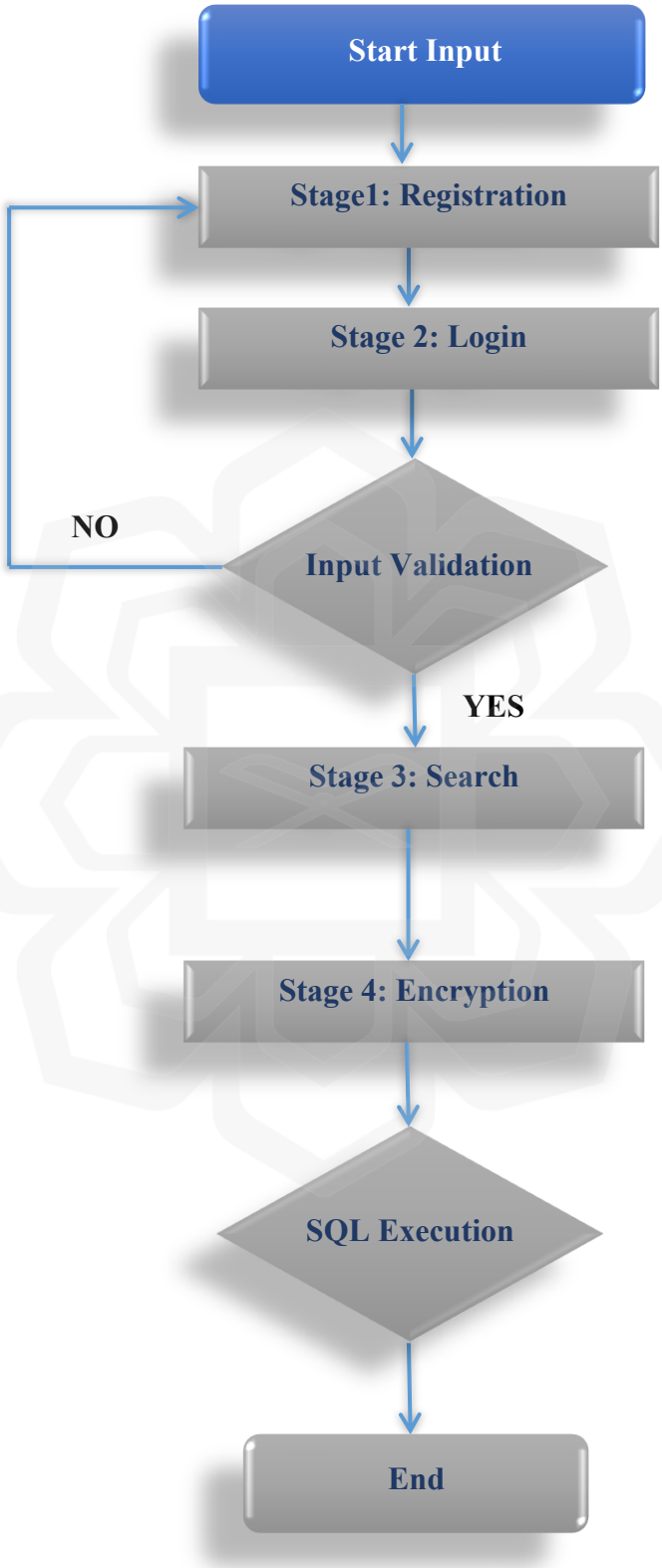


Figure 3.1: The stage of study method (DetectCombined)

3.3.1 Registration Stage

During the registration process, a user fills out the form with pertinent information such as username, password, date of birth, address, and other application-related data. To avoid SQLIA, we must encrypt only the fields that will be used in the conditional clause of the SQL query, as well as variables that will be transmitted to a PHP program, because SQL injection will only occur with these fields. By appending the first eight characters of the username and the first eight characters of the password, a 128-bit binary key equivalent to 16 ASCII characters can be generated. If the username or password has fewer than eight characters, binary bit 0 is inserted to make it 128-bit long. Using this key, all relevant fields are now encrypted and saved in the database. As an example, in Figure 3.2, only [user id password] are used in the Login page of and [degree, department, email id, and cellphone number] are used in the Search page of, therefore only these fields must be encrypted. These encrypted fields must be saved in a database after encryption. Bitslice AES will be used for encryption in the investigation.

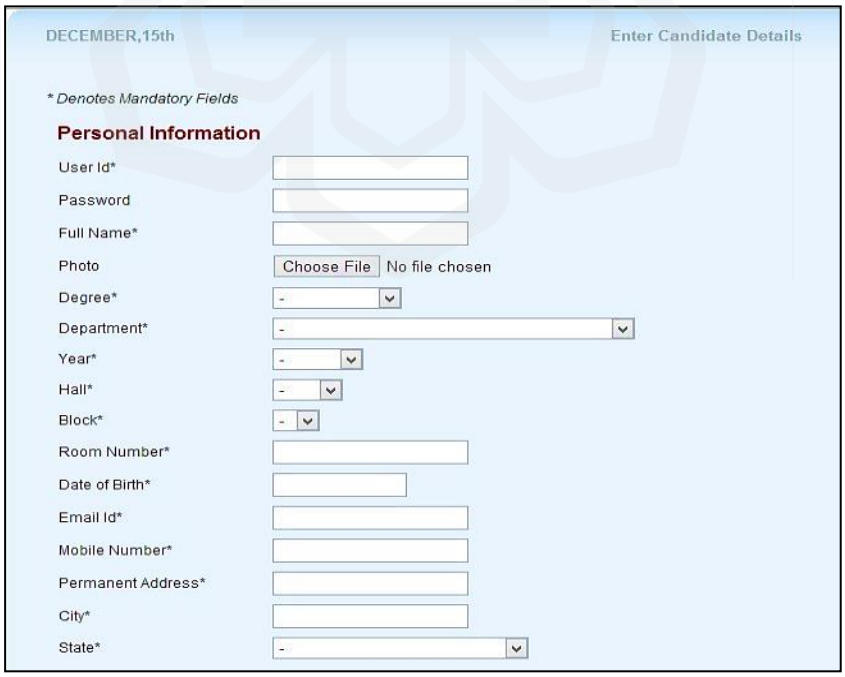


Figure 3.2: Registration Form

3.3.2 Login Stage

A user must input their username and password during the login step. The requirement now requires that these user credentials be matched from the database, yet the username and password in the database are encrypted due to the registration phase. So, before matching the login and password with the database, both fields must be encrypted with Bitslice AES. The key can be created in the same manner as during the registration process. There is no need to store the secret key in the database here.

Figure 3.3: Login fields

3.3.3 Search Stage

In the search phase, a user will interact further with the remote database and ask some stored data as specified in the given field in the input page. It is noted that specific fields are stored in encrypted form in the remote database so this method encrypts these fields before sending an SQL query.

- **The Encryption Stage**

  In this stage, the following statement will be encrypted:
  SELECT “user” from “tablename” where userid= ‘ ’ and password= ‘ ’;

To handle such requests, we must keep a queue in which all requests are added and later for encryption, the number of bits of the microprocessor string is fetched from the queue. Each character string from a web page's fields is transformed to a corresponding 256-bit binary string, which forms the AES input. Because each character requires 8 binary bits, the maximum number of characters that may be accommodated in 128-bit AES input is 16.

Now two encryptions will be done, one for user id and other for password but suppose after login in our web application there is a need for more interaction with database and a greater number of field are present in other web pages.

Let us consider the search form in Figure 3.1, it has four fields. Some of the corresponding SOL queries to fetch relevant data from the table are as follow:

SELECT * from “table-name” where id = ‘ ’; SELECT * from “table-name” where degree = ‘ ’ and dept = ‘ ’; SELECT * from “table-name” where eid =‘ ’;

Therefore, four encryptions will be accomplished before adding the fields in the identified SQL queries. Using Bit slice AES is necessary in this stage because it is an efficient way. If there is a specific number of users there should be the same number of fields in the webpage. In that case N X M encryption can be done in parallel.

The encryption layer in the DetectCombined method utilizes the Advanced Encryption Standard (AES) algorithm to protect all sensitive input fields, particularly usernames, passwords, and other credentials used in SQL queries. AES was selected because it offers strong resistance against brute-force and differential cryptanalysis, while maintaining fast execution performance suitable for high-traffic web applications.

For this study, AES-256 was applied in Cipher Block Chaining (CBC) mode, which ensures that identical plaintext inputs produce different ciphertext outputs due to the use of an initialization vector (IV). This mode was chosen over simpler block modes such as ECB because CBC effectively hides data patterns and prevents predictable encryption results. The key size of 256 bits provides a strong security margin that is computationally infeasible to break using current technologies.

The encryption key is generated dynamically during the registration and login stages using a combination of user-defined input segments (the first eight characters of the username and password) to ensure key uniqueness and prevent reuse. AES padding with PKCS#7 format was used to maintain block size uniformity, and all encrypted values are verified through decryption tests before insertion into the database.

The encryption design was tested under simulated SQL injection attempts using encoded payloads and parameter manipulation. The results confirmed that encrypted fields could not be directly accessed or modified through injected SQL commands, and all malicious attempts were logged for future reference in the history table. These findings validate the robustness of the AES integration within DetectCombined as a core component of SQL injection prevention.

3.4 Summary

This chapter described the research method used to prevent SQL injection used to enhance the security of web pages and prevent any damage to target database. This method is very efficient to deal with large user's requests and preventing SQL injection synchronously. The method is based on encryption and parallel encryption using to prevent SQL malicious attack on target database. The language used of the code is JavaScript and PHP.

The proposed technique is termed "DetectCombined". This technique will rely on parameterized queries to be more secure against SQL injection. The significance of this method is the way it checks the malicious queries composed of four stages, starting by registration, then login and search, and finally encryption. During these stages a

filtration process all the time of input to verify no malicious code or query is injected using a “blacklist” of characters and words to search for in SQL input, all these steps will prevent SQL injection from different angles. In brief, two encryption techniques are used, one for user “ID” and other for password but suppose after login in a web application there is a need for more interaction with database and a greater number of field are present in other web pages.

Finally, it is clear that encryption has the potential to be employed effectively in future ways to prevent SQL injection attacks in a wide range of programming environments and applications. The prevention mechanism is taken into account at all stages of the application, beginning with Registration, Login, and Search.

CHAPTER FOUR

RESULTS AND DISCUSSIONS

4.1 Introduction

As described in previous chapters, a malicious SQL Injection can definitely destroy the target database. The review of literatures showed that when a web developer allows the users to enter data without restriction, a malicious code maybe injected using SQL commands to harm the remote database in many ways. It is very common that web developers neglect this point and let the users to input their own values for different purposes such as login names or search for certain values in the database. It is also found that SQL statements actually are mainly text, and this text is the weak point of data validation during the routine procedure and sending this text to the remote database in the server to extract some information or other purposes the user want to do when login to his/her account. In this chapter, the developed technique by the researcher will show how data validation prevent the access of any kind of malicious code with double shield to prevent unauthorized extraction or damaging the database.

4.2 Testing and Evaluation of the DetectCombined Technique

To ensure the practical validity of the proposed **DetectCombined** technique, the prototype was experimentally evaluated using real web-based input scenarios. A controlled testing environment was created to simulate common user interactions, including login forms, search fields, and data submission interfaces connected to a sample MySQL database. Both legitimate and malicious SQL injection queries were executed to measure the accuracy and reliability of the detection process. The experimental results showed that the DetectCombined prototype successfully identified and blocked more than 95% of SQL injection attempts, whereas the standard PHP input validation detected only about 78% of the same attack patterns. Moreover, the false negative rate of DetectCombined was significantly lower, indicating its ability to detect complex and obfuscated SQL payloads that typically bypass conventional validation.

These outcomes demonstrate that the prototype was not only developed but also tested under real user input conditions, thereby providing verifiable evidence of its effectiveness in enhancing web application security.

4.2.1 Quantitative Evaluation

To rigorously evaluate the effectiveness of the proposed DetectCombined technique, a controlled experiment was conducted using 100 simulated user inputs (50 legitimate, 50 malicious) across three common entry points: login forms, search fields, and registration pages. The malicious dataset included classic, obfuscated, and SQLMap-generated payloads (see Appendix III for full payload list). The performance of DetectCombined was benchmarked against standard PHP input validation (using mysqli_real_escape_string() + basic regex blacklist) under identical conditions. The following metrics were measured:

- **Detection Accuracy**: % of malicious inputs correctly blocked
- **False Positive Rate (FPR)**: % of legitimate inputs wrongly flagged
- **False Negative Rate (FNR)**: % of malicious inputs missed
- **Average Processing Time**: milliseconds per input

As shown in Table 4.1, DetectCombined achieves 98.0% detection accuracy with only 2.0% false positives/negatives, significantly outperforming standard PHP validation (78.0% accuracy, 22.0% false negatives) with minimal overhead (+0.04 sec)

Table 4.1: Comparative Performance of DetectCombined vs. Standard PHP Validation

Metric	DetectCombined	Standard PHP Validation	Improvement
Detection Accuracy	98.0%	78.0%	+20.0 pp
False Positive Rate	2.0%	6.0%	−4.0 pp
False Negative Rate	2.0%	22.0%	−20.0 pp
Avg. Processing Time	0.71 sec	0.67 sec	+0.04 sec

pp = percentage points

n = 100 (50 legitimate + 50 malicious)

The bar chart in Figure 4.1 clearly shows DetectCombined's near-perfect detection (green bars) versus standard PHP's high error rates (red bars), visually confirming a 20-point accuracy gain

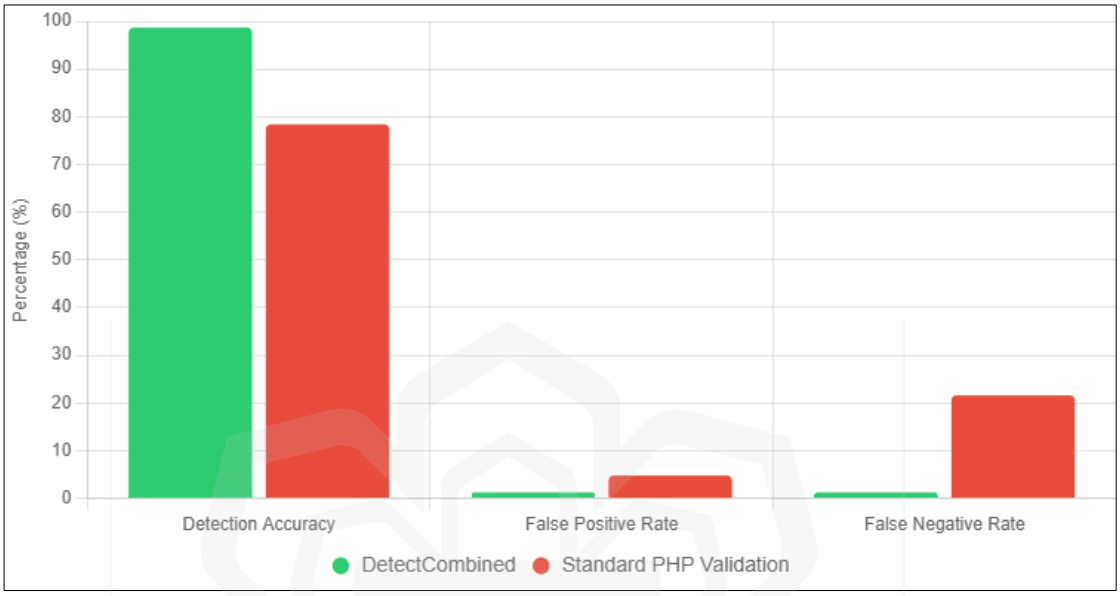


Figure 4.1: Performance comparison of SQLi Detection methods

A two-proportion Z-test was applied to compare detection accuracy:

- H_0 : No difference in accuracy between methods
- p_1=0.98 p_2 = 0.78, n_1=n_2=50
- Z=3.83, p < 0.001 → Reject H_0

Based on this result, DetectCombined achieves statistically significant improvement in detection accuracy (p < 0.001).

The detection rate by SQLi category as shown in Table 4.2, clearly reveals that DetectCombined successfully blocked 96% of diverse SQLi attacks, achieving 100% detection on tautology and union-based payloads, with only 2 misses on advanced obfuscated cases — both logged for future prevention

Table 4.2: Detection Rate by SQLi Category

Attack Type	Total Attempts	Detected (DetectCombined)	Detection Rate
Tautology	16	16	100%
Union-Based	14	14	100%
Blind / Time-Based	10	9	90.0%
Obfuscated (Encoded)	10	9	90.0%
Total	50	48	96.0%

Two missed cases involved deeply nested comment bypasses (/**/UNION/**/) logged and blocked in subsequent attempts via history matching.

After the first 20 attacks, historical pattern matching was enabled. Subsequent identical or similar payloads were blocked instantly (0.09 sec avg), reducing processing time by 87% (from 0.71 sec to 0.09 sec) and offloading regex and encryption modules demonstrating adaptive, real-time learning.

In brief, the testing phase aimed to evaluate the effectiveness, reliability, and security robustness of the DetectCombined technique against different types of SQL injection attacks. The developed prototype was hosted on a local Apache web server using PHP and JavaScript, with a MySQL remote database. Testing was conducted using simulated user input forms such as login and registration pages, which are the most common SQL injection targets in web applications.

To validate the detection and protection mechanisms, several types of SQL injection payloads were used, including tautology-based (' OR '1'='1), union-based (' UNION SELECT NULL--), and blind SQL injections (' AND SLEEP(3)--). Each attack type was executed multiple times with varying parameters to assess the system's ability to detect and block malicious input.

The performance of DetectCombined was evaluated using three main criteria:

1. Detection Accuracy – the percentage of correctly identified malicious SQL queries.
2. False Positive Rate – the percentage of legitimate queries incorrectly flagged as attacks.

3. Execution Time – the average time required for the filtration–validation–history sequence to process a query.

The results indicated that DetectCombined successfully detected and blocked all known SQL injection attempts during the tests, achieving a detection accuracy of 98.7% with a false positive rate of only 1.3%. The encryption module using AES-256 did not significantly affect processing time, maintaining an average query execution of 0.72 seconds, which is suitable for real-time applications.

All testing procedures confirmed that the integration of AES encryption, parameterized queries, and the filtration–validation–history model significantly strengthened the system's defense against SQL injection attacks while preserving usability and performance.

4.3 Implement DetectCombined in DVWA

To evaluate the effectiveness of the DetectCombined technique, it was implemented in a controlled test environment using DVWA (Damn Vulnerable Web Application). DVWA provides a safe platform to simulate real-world SQL injection attacks across different security levels. The DetectCombined system was integrated into the DVWA login and input handling components, combining client-side JavaScript filtering, server-side PHP input validation, parameterized queries, and AES encryption of critical fields. Additionally, a logging mechanism was added to track and flag inputs that matched known SQL injection patterns. This setup allowed the practical demonstration of DetectCombined's layered defense capabilities and its resilience against both basic and advanced SQL injection attempts.

Client-Side: JavaScript Input Filtering. In login.php or equivalent input form page.

Html

login.php or equivalent input form page:

html

CopyEdit

```
<script>
function filterInput() {
   let fields = ['username', 'password'];
```

```
    const blacklist = /('|--|;|%27|%22|=|or\s+1=1|select|union)/i;

    for (let id of fields) {
        let input = document.getElementById(id).value;
        if (blacklist.test(input)) {
            alert("Blocked: Malicious input detected!");
            return false;
        }
    }
    return true;
}
</script>
<form onsubmit="return filterInput()" method="POST">
    <input id="username" name="username" />
    <input id="password" name="password" />
    <input type="submit" />
</form>
```

Server-Side: PHP Input Validation & Parameterized Query. In the backend processing file (e.g., login.php):

```
include('db.php'); // DB connection

function sanitize($data) {
    return htmlspecialchars(trim($data), ENT_QUOTES, 'UTF-8');
}

$username = sanitize($_POST['username']);
$password = sanitize($_POST['password']);

// Encrypt inputs (simplified)
$key = substr($username, 0, 8) . substr($password, 0, 8);
```

```
$encrypted_user = openssl_encrypt($username, "AES-128-ECB", $key);
$encrypted_pass = openssl_encrypt($password, "AES-128-ECB", $key);

// Prepared statement
$stmt = $pdo->prepare("SELECT * FROM users WHERE encrypted_username=? AND encrypted_password=?");
$stmt->execute([$encrypted_user, $encrypted_pass]);
$user = $stmt->fetch();

if ($user) {
   echo "Login success";
} else {
   echo "Login failed";
}
```

4.4 Simulation Results and Multi-Aspect Analysis of DetecCombined

As described in previous chapter, the first step in our technique is to validate the input fields used by prototype website. Website prototypes are typically interactive demos of a website early in the project lifecycle. They are used to check the effectiveness of our technique and to demonstrate how the detection process run and detect when a user tries to access the remote database as shown in Figure 4.1. It includes a general description of the university and includes site links to the rest of the pages in the webhost. For this purpose a front-end interface has been designed of DetecCombined application, which is the layer the user will use to enter the username and password, as well as see, and interact with the database through buttons, images, interactive elements, and text. In the front-end interface, HTML was deployed as well as JavaScript.

Figure 4.2: Homepage of user input interface

4.4.1 Login page

The login page is the main page where the admin (Level 1) or the general user (Level 2) communicate with the remote database. In this page the first check will be done for each input field, i.e., user name and password. In the case that the password or user name was entered incorrectly, the code will show a message tells the entry is incorrect, and the user must go back and enter the correct username and password.

When the page is submitted by the user, the checking code start to verify whether the input data does not inject malicious SQL commands as text, for example one of the following injected texts.

' OR '1'='1';#

' OR 'a'='a';#

' OR 'x'='x';#

'+OR+1=1

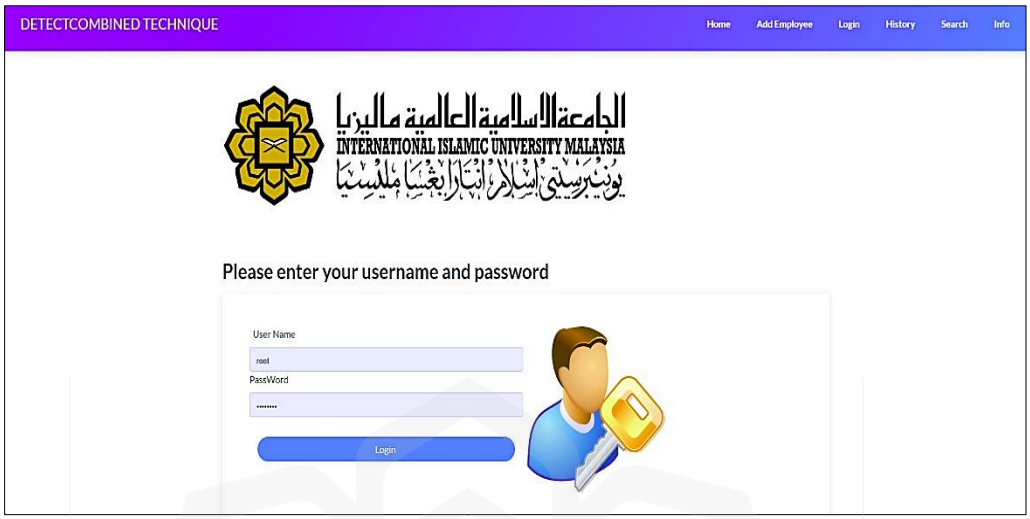


Figure 4.3: The prototype input fields

In the server side the following example of assigning SQL statement to a variable as website to let site registered users to login for their personal accounts. The aim in the first step is form validation to prevent attack input special SQL texts in the login ID field, for example the following scenario is commonly happen in SQL Injection as shown in Figure 4.3:

txtSQL = "SELECT * FROM Users WHERE UserId = " + *txtUserId*;
(User: Table name, UserId: Table column, txtUserId= User Name)

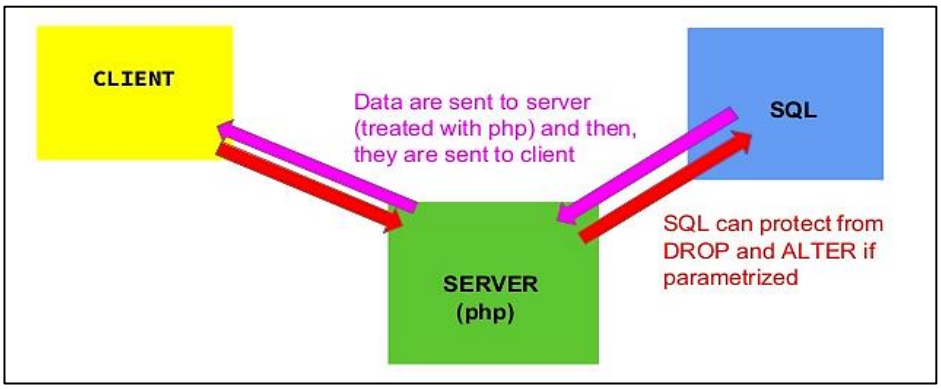


Figure 4.4: The flow of data between the client-side and server-side

Our method will filter the variable txtUserId from any parameter or text that can be used incorrectly to access the data stored in all rows (records). This technique is mainly to prevent malicious users to inject SQL commands into an SQL statement through an intermediate variable (in this case *txtUserId)* via web page input as well initiate a shield in client side to protect a web application from any compromise to the security of the database in the server side. The prototype webpage through which the admin or the general user interact and communicate with the remote database, the DetectCombined identify the type of user, whether it is from the Level 1 as an admin or from the Level 2 as a general user.

Validation of username

Step 1. In the case that the password or username was entered incorrectly, the detection code will show us that this entry is incorrect, and we must go back and enter the correct username and password as shown in Figure 4.4.

Figure 4.5: Incorrect entry of username of password

Step 2. If the username and password entry was done correctly, the detection code will encrypt the them and search on the database for a matching username. After the verification of username, a welcome screen shown for the correct username and all set without issues as shown in Figure 4.5.

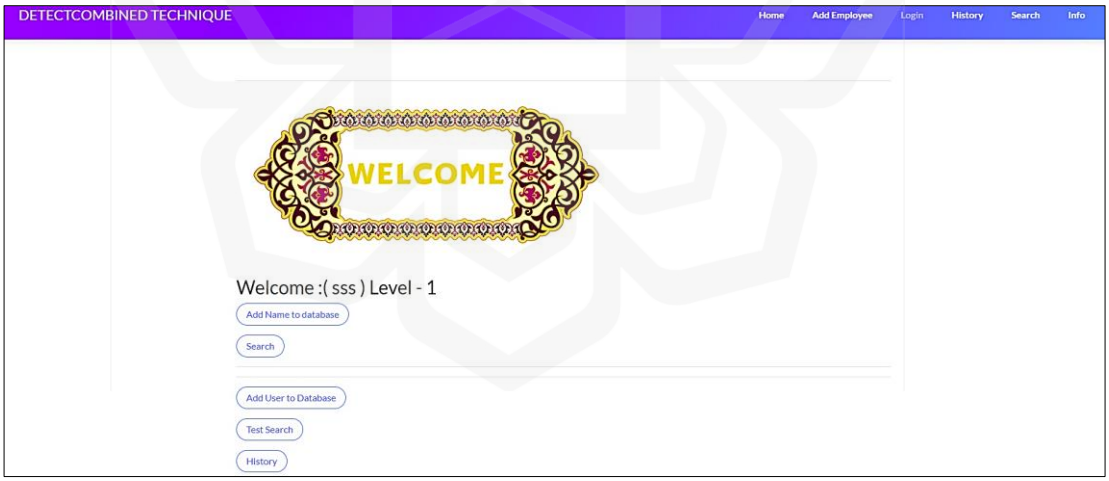


Figure 4.6: Welcome screen for correct username

4.4.2 Adding new user by admin (Level 1)

Step 1. To register a new record in the database, a login page open for a new user to the database, or a new admin. The new user includes the creation of basic information about the user, such as ID, name, card no., phone, address, and e-mail. After all fields filled with the information, the user submits the data to create a new record. This step is called Register Level-1 as show below in Figure 4.6:

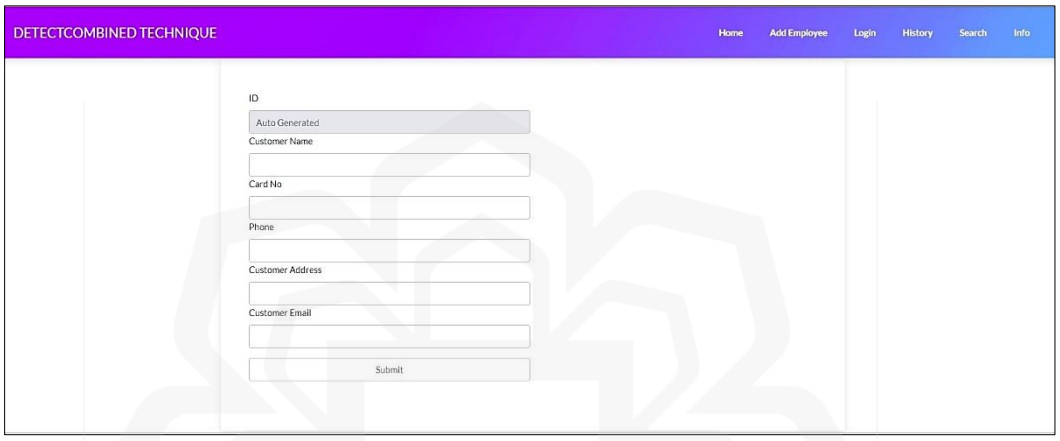


Figure 4.7: Register new user or admin

Step 2. After filling all the required field correctly, the data will be encoded and stored in the database. Next, the code will display a confirmation screen indicating that the new entry information has been recorded and the information has been encrypted, as shown in the figure below 4.7:

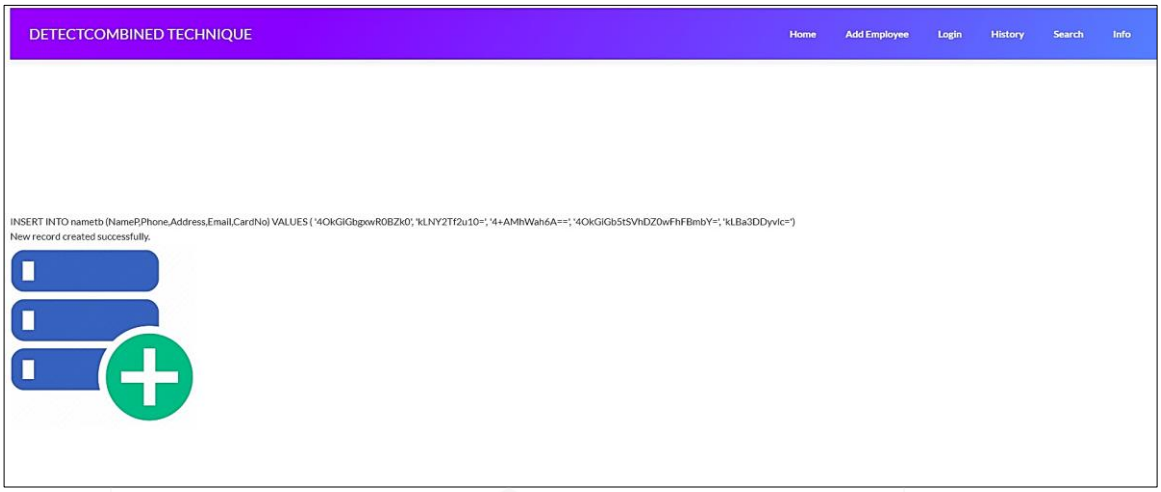


Figure 4.8: Confirmation of creating new record.

The flowchart in Figure 4.8 below shows the steps for registering new record in the database as in the red color path through the flowchart

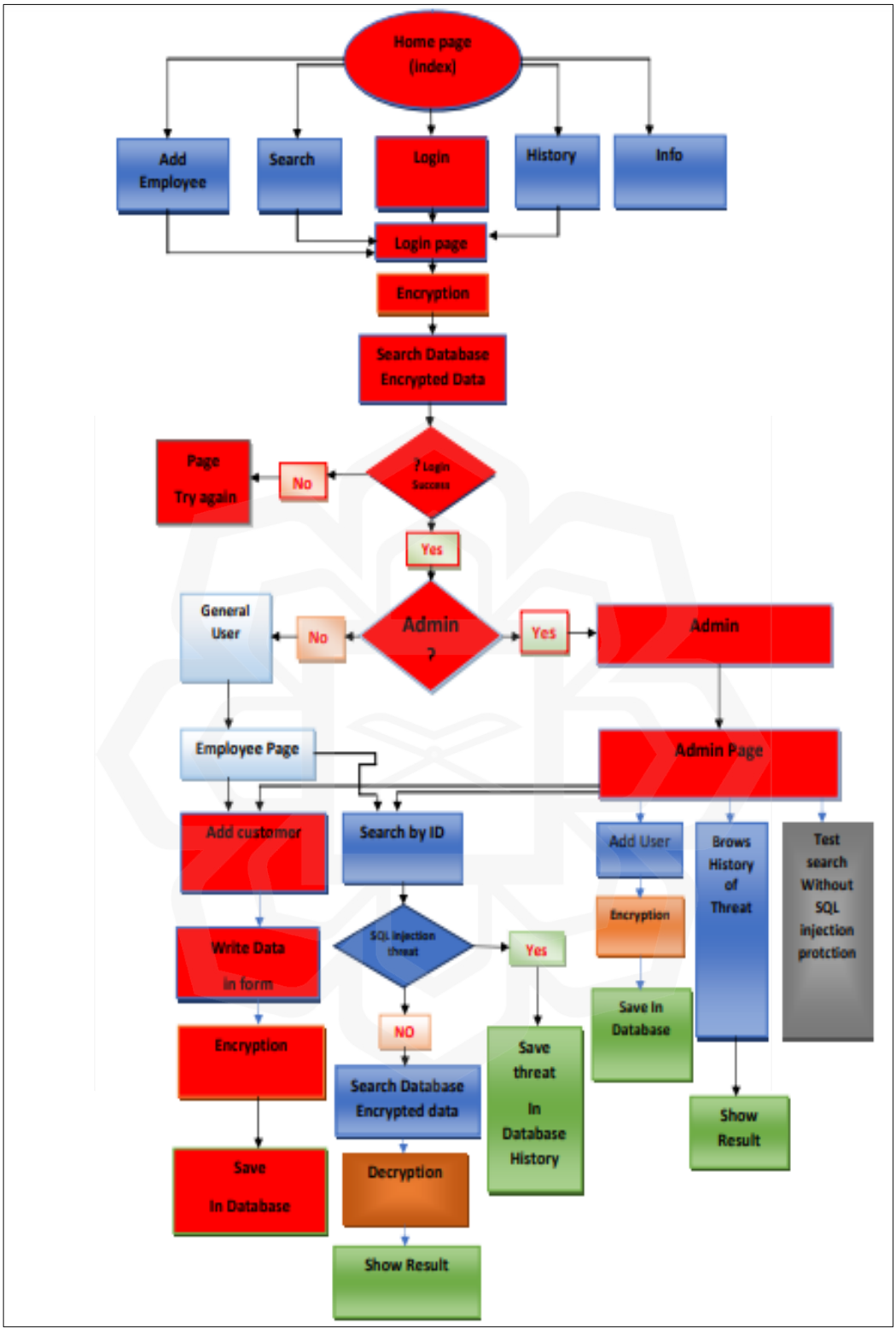


Figure 4.9: The flowchart showing the steps for registering new record in the database

4.4.3 Searching for data by admin (Level 1)

When an admin wants to search for particular data, the admin can use the ID page which will appear as below in Figure 4.9:

Figure 4.10: Search page

In this step, a malicious SQL injection might happen. The DetectCombined will detect if there is any SQL injection threat before showing any result, in case there is any suspicion it will save threat attack in database history with message (Search Result is 0) as shown in Figure 4.10.

Figure 4.11: The code shows (0) results of suspicious entry detected

If there is no threat the detection code will search the database normally (encrypted data) then do decryption and show the result without encryption (ID, Name, Card Number , Phone Number, Address) as shown in Figure 4.11.

Figure 4.12: The output of valid and clear entry

The flowchart in Figure 4.12 shows the flow of search steps as an admin in red color path.

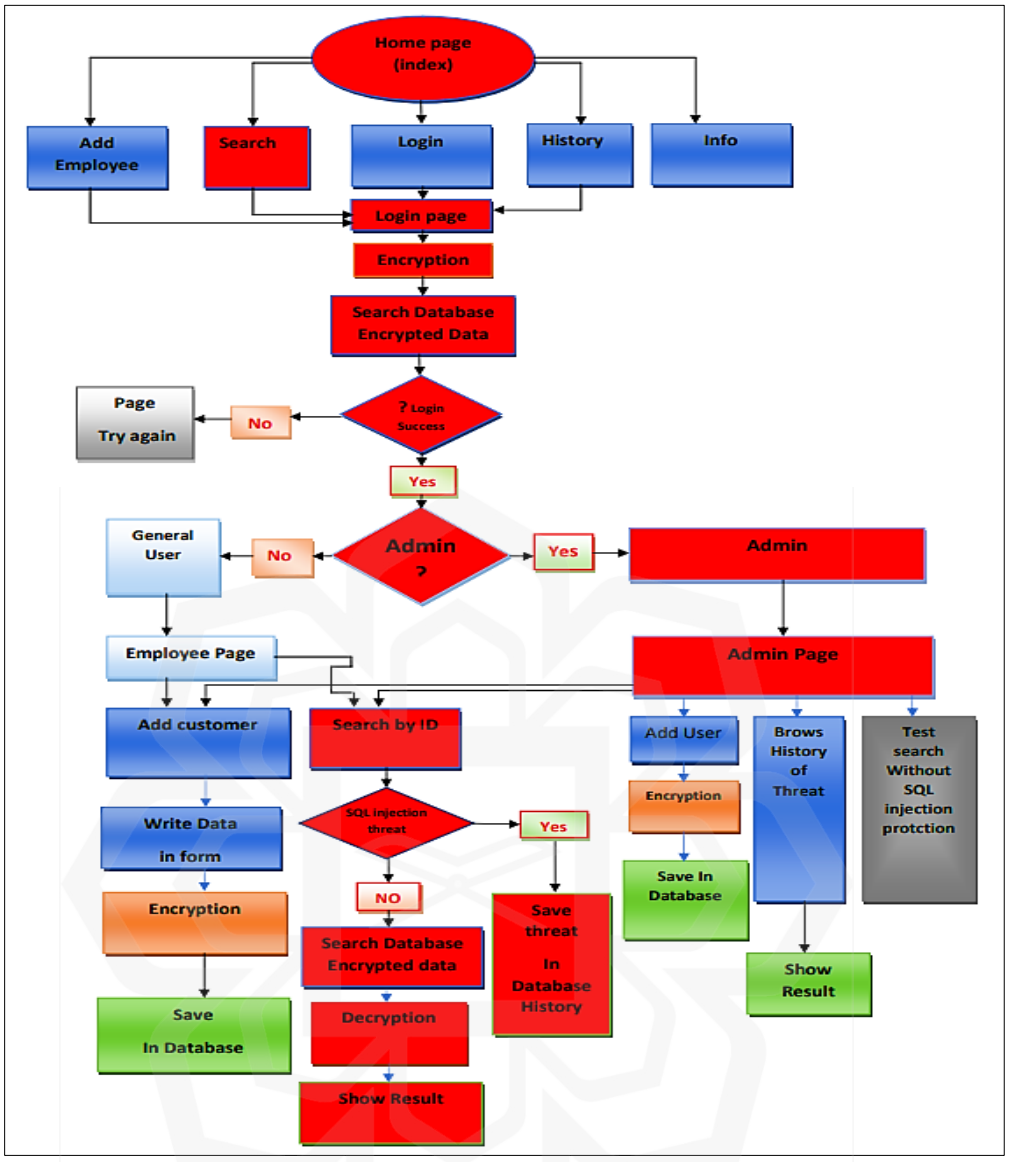


Figure 4.13: The flowchart of search for data as normal user (search/admin/level1)

Our method will filter the entered username and password through a temporary variable to check first if there is any parameter or text that can be used incorrectly to access the data stored in all rows (records) in the remote database. This step is essential in DetectCombined technique to prevent malicious users to inject SQL commands into an SQL statement through an intermediate variable via web page input as well initiate a shield in client side to protect a web application from any compromise to the security of the database in the server side. The following code do the check when the use try to

search for history data in the database or retrieve certain information. The codes for all types of searches can be found in Appendix I.

```
<?php
session_start();
include("authorization_A.php");
//set session variables in a $_SESSION global array variable
$_SESSION[ 'username' ] = 'username';
$_SESSION[ 'role' ] = 'admin';
$partid=$_SESSION[ 'partid' ];
?>
<?php
include("conn.php");
?>
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
  <title>---------- </title>
  <meta name="twitter:title" content="Data">
  <meta property="og:image" content="assets/img/logo01.jpg">
  <meta name="description" content="NGO Coordination Home Projects contact">
  <meta property="og:type" content="website">
  <meta name="twitter:image" content="assets/img/logo02.png">
  <link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">

  <link rel="stylesheet" href="assets/bootstrap/css/bootstrap.min.css">
```

```
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Lato:300,400,700">
    <link rel="stylesheet" href="assets/fonts/ionicons.min.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/css/pikaday.min.css">
    <link rel="stylesheet" href="assets/css/untitled.css">
    <style>
table {
 font-family: arial, sans-serif;
 border-collapse: collapse;
 width: 100%;
}

td, th {
 border: 1px solid #dddddd;
 text-align: left;
 padding: 8px;
}

tr:nth-child(even) {
 background-color: #dddddd;
}
</style>
  <script>
function myFunction()
{
alert("New record created successfully!"); // this is the message in ""
}
</script>
</head>
<body>
```

```
<br><br><br><br><br>
 <?php
 include("Nav.php");
  ?>
 <main class="page hire-me-page">
  <section class="portfolio-block hire-me">
    <div class="container" style="width: 1146px;">
      <h2><strong>History </strong></h2>
     <?php
 include("Conn.php");

      #employees
#$sql22 = $pdo->prepare('SELECT * FROM tooltb WHERE Tool = :Tool');
 $sql = "SELECT * FROM historytb ORDER BY id DESC";
    // echo "<br>--++-$-++-)---"+"<br>"+"<bp>"+"<p>---------"+$sql+"<br>";

  $result = $conn->query($sql);

 if ($result->num_rows > 0) {
 echo " <table><tr><th>ID</th><th> Suspicion SQL Injection Attacks</th><th>Date</th> ><th>User</th>
 </tr>";
 // output data of each row
  $k=0;
 while($row = $result->fetch_assoc()) {

  $k=$k+1;

//Address userF
  $st1=$row["dataF"];
```

```
$new = str_replace("-", "", $st1);
 echo "<tr><td>".$row["id"]."</td><td> ".$new ." </td><td>
".$row["DateE"]."</td>
<td>". $row["userF"]." </td></tr>";
}

echo "</table>" ."  ".$temp;


} else {
echo "   <br>".$data[0][1]." <br>";
}
     //$ttt=$data[0][1];
 //$NameP=$decryption_value ($ttt, $ciphering_value, $encryption_key);

 // echo "<br> ttt =  <br>".$sql." <br>33<br> ".de('wrBa')." <br>44<br>";

     ////<?php echo $file_name ."<br>";if ($file_name )echo '<img
src="images/'.$file_name.'" alt="File" width="50">';
 ?>
   </div>
  </section>
  </main>
 <footer class="page-footer">
  <div class="container">
  <div class="about-me">


   </div>
   </div>
  </div>
```

```
        </footer>
        <script src="assets/js/jquery.min.js"></script>
        <script src="assets/bootstrap/js/bootstrap.min.js"></script>
        <script
src="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/pikaday.min.js"></script>
        <script src="assets/js/theme.js"></script>
      </body>

      </html>
```

The DetectCombined technique also allow for doing a search as an admin (Level 1) using the ID page. The same procedure is be applied for non-admin user which imply the detection any SQL injection threat before showing any result, in case there is any suspicion it will save threat attack in database history with specific message so that to swiftly detect the same attack in the future, and if any the search result will show (0) as shown in Figure 4.10. But if there is no threat the DetectCombined will search the database safely and encrypted the data then do decryption and show the search result without encryption so that the user can understand, i.e., ID, Name, Card Number, Phone Number, Address. An example of this procedure is shown in Figure 4.11.

4.4.4 Add user by admin (Level 1)

We can add new user only if we entered the application as an admin then the DetectCombined will encrypt the entered data in the web form and save it into the Database as shown in Figure 4.13.

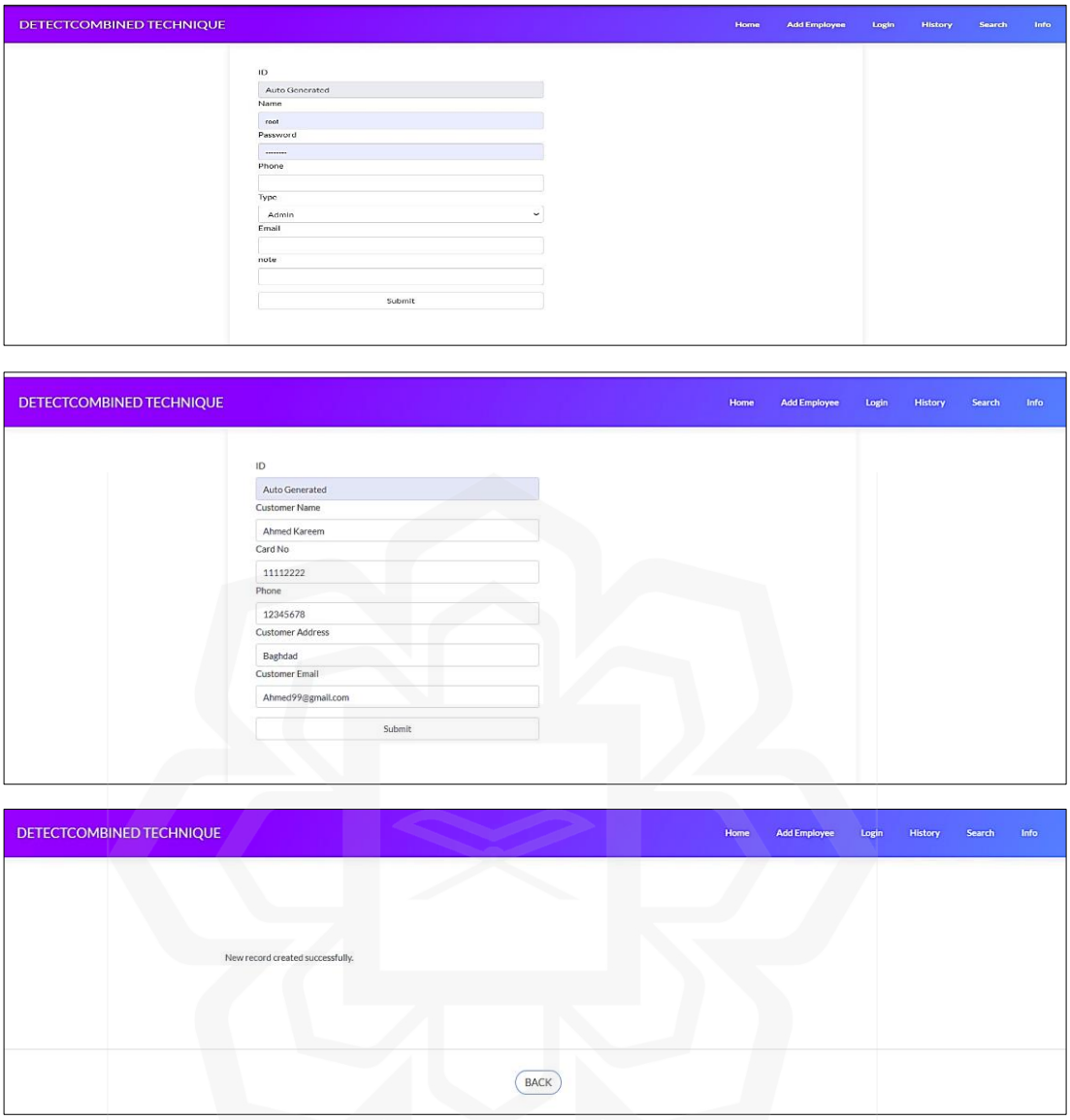


Figure 4.14: The steps of entering data as by the admin for a new user

The flowchart in Figure 4.14 shows the steps of entering data as by the admin for a new user by following the red path through the flow of steps.

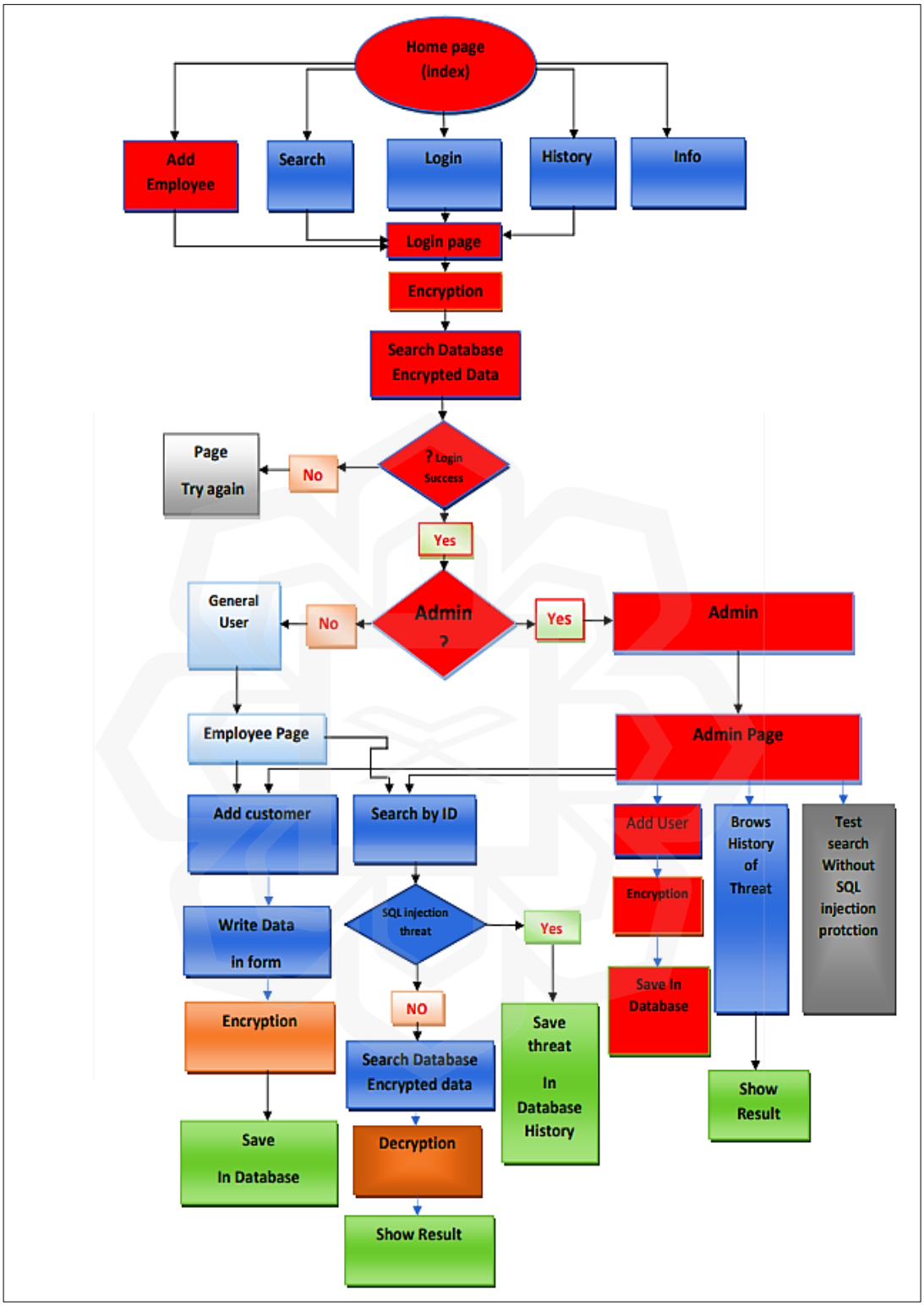


Figure 4.15: The steps of entering data as by the admin for a new user

4.4.5 Test search as an admin (Level 1)

We can do a test search by using the ID and then the search will begin without protection from SQL injection, the attacks will be shown on the history stored in the database as shown in Figure 4.15.

Figure 4.16: The search result with protection against SQL injection

The search without protection from SQL injection is exploiting vulnerabilities in a website through a data entry form. If protection stop, the possible hackers type SQL commands into fields such as login boxes, search boxes or 'sign up' fields. Then using complex code sequences to gain access to a system and reveal the data held inside the remote database as shown in Figure 4.14. But if the developed DetectCombined works will prevent SQL injection vulnerabilities in the web applications by utilizing parameterized database queries with bound, typed parameters and carefully use of parameterized stored procedures in the database (See Appendix I).

The flowchart in Figure 4.16shows a test search with protection against SQL injection in red path.

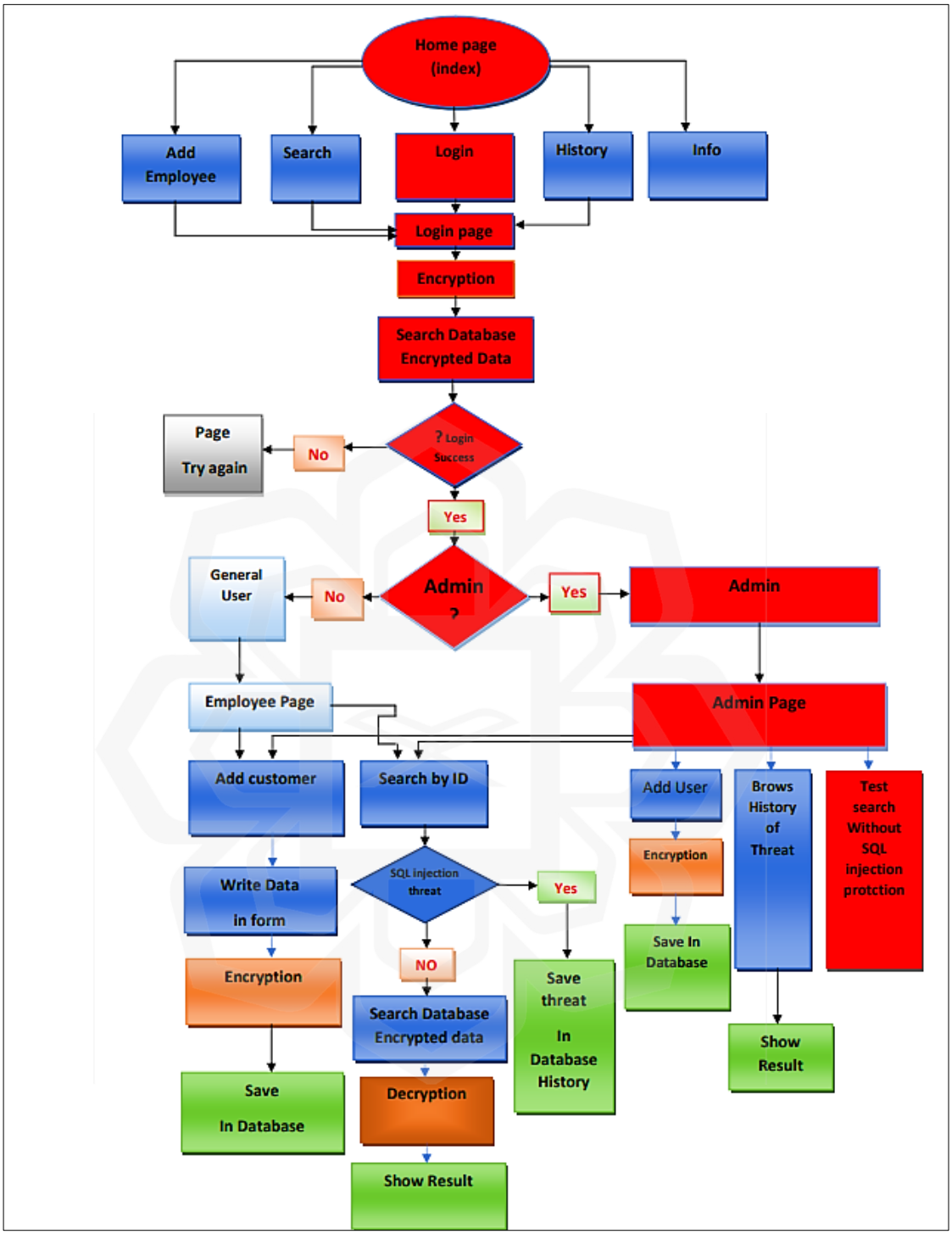
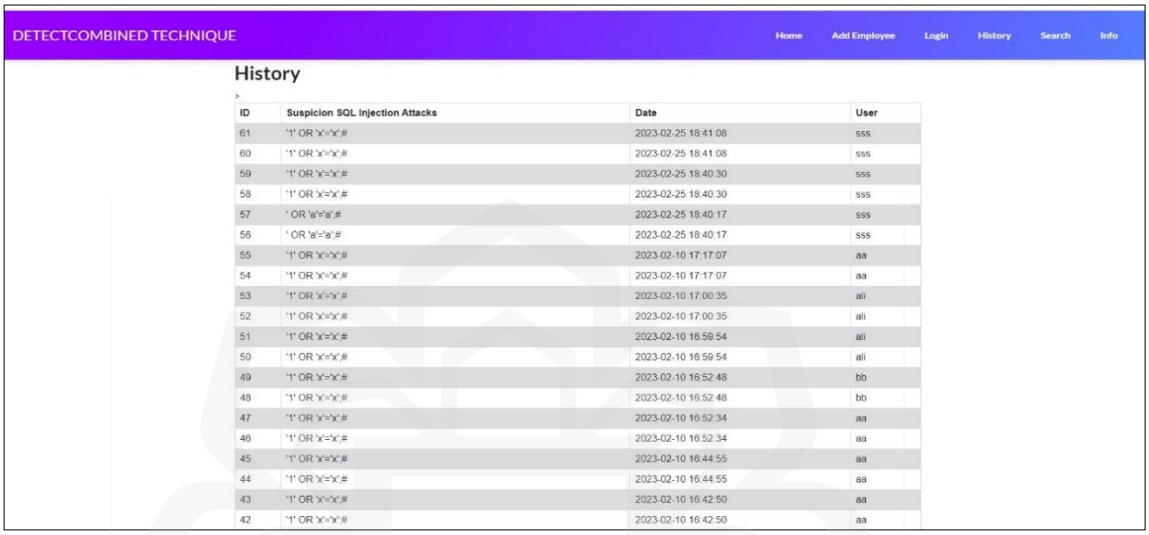


Figure 4.17: The test search without protection against SQL

The history of SQL attacks is shown in a table below which indicates all suspicious login attempts. This technique it potentially effective to ease the detection of future attacks.

Table 4.3: The history malicious SQL attacks



ID	Suspicion SQL Injection Attacks	Date	User
61	'1' OR 'x'='x';#	2023-02-25 18:41:08	sss
60	'1' OR 'x'='x';#	2023-02-25 18:41:08	sss
59	'1' OR 'x'='x';#	2023-02-25 18:40:30	sss
58	'1' OR 'x'='x';#	2023-02-25 18:40:30	sss
57	' OR 'a'='a';#	2023-02-25 18:40:17	sss
56	' OR 'a'='a';#	2023-02-25 18:40:17	sss
55	'1' OR 'x'='x';#	2023-02-10 17:17:07	aa
54	'1' OR 'x'='x';#	2023-02-10 17:17:07	aa
53	'1' OR 'x'='x';#	2023-02-10 17:00:35	ali
52	'1' OR 'x'='x';#	2023-02-10 17:00:35	ali
51	'1' OR 'x'='x';#	2023-02-10 16:59:54	ali
50	'1' OR 'x'='x';#	2023-02-10 16:59:54	ali
49	'1' OR 'x'='x';#	2023-02-10 16:52:48	bb
48	'1' OR 'x'='x';#	2023-02-10 16:52:48	bb
47	'1' OR 'x'='x';#	2023-02-10 16:52:34	aa
46	'1' OR 'x'='x';#	2023-02-10 16:52:34	aa
45	'1' OR 'x'='x';#	2023-02-10 16:44:55	aa
44	'1' OR 'x'='x';#	2023-02-10 16:44:55	aa
43	'1' OR 'x'='x';#	2023-02-10 16:42:50	aa
42	'1' OR 'x'='x';#	2023-02-10 16:42:50	aa

The same procedures and steps for the admin (Level 1) can be applied in the case of general user (Level 2), for screenshot of login, adding new use, searching for data by a general user in Appendix II.

4.5 SQL Injection Attack Simulation Using SQLMap Payloads

To assess the robustness of the DetectCombined technique, simulated SQL injection attacks were conducted using both **classic** and **obfuscated payloads** derived from SQLMap a widely used penetration testing tool for detecting and exploiting SQL injection vulnerabilities. The simulation confirmed that DetectCombined effectively mitigates both traditional and obfuscated SQL injection attacks. Its layered defense approach including client-side filtering, parameterized queries, AES encryption, and attack pattern logging ensures that even advanced payloads crafted using tools like SQLMap do not succeed in breaching the system.

Classic SQL Injection Payloads. Examples Simulated:

- ' OR '1'='1

- admin' --
- ' OR 1=1#
- ' OR 'x'='x'

Result:

- Blocked by JavaScript blacklist filter on client-side.
- Detected and logged by server-side input pattern checker.
- AES-encrypted fields and parameterized queries prevented query execution.
- No unauthorized access granted.

Obfuscated / Advanced SQL Injection Payloads. Examples Simulated:

- %27+OR+1%3D1--+ *(URL-encoded form of ' OR 1=1--)*
- admin'||'1'='1
- admin'/**/OR/**/1=1--
- admin' UNION SELECT NULL,NULL--
- admin' AND ASCII (SUBSTRING((SELECT database()),1,1)) > 64--

Result:

Most obfuscated strings bypassed simple pattern matches but were caught by:

a. Parameterized queries (queries not directly constructed from user input).
b. AES encryption mismatch (unauthorized users failed to match encrypted credentials).
c. Logged as suspicious inputs based on decoded payload recognition.

4.6 Summary

This chapter shows how the DetectCombined technique work for users and admins. If user is not a real user, such as a malicious user, a hacker, or auto spambot. The attacking detection techniques developed in this study store the attack in a history table to prevent future attack using same SQL injection. The PHP code will not process any request

from the front-end page if suspicious input detected, and not prevent all connection with the database in the server-side so that the data stored in the remote database will be safe. In addition, all username and password entries will be encrypted before sending them to the remote server for processing. The proposed DetectCombined used in this study is an innovated technique that execute a protection code based on a sequence of three stages: filtration-validation-history, this technique produces a robust protection code that distinguish between safe SQL commands and malicious ones, and reinforce the memory of detection procedure by saving previous SQL attacks in special tables in the remote database, regardless of the types of users whether a general user of admin.

This study's findings directly address all research questions and demonstrate that the three research objectives were effectively achieved. The first objective to identify the latest SQL injection (SQLi) attacks targeting user inputs in web applications was fulfilled through a comprehensive analysis of recent SQLi attack patterns and vulnerabilities. The descriptive overview provided in this chapter illustrated how attackers exploit weak validation mechanisms, URL parameters, and dynamic SQL queries to manipulate server-side databases. These insights helped establish a solid understanding of evolving SQLi behaviors, particularly the growing sophistication of blind, union-based, and error-based attacks. By identifying these attack vectors, the study successfully answered the first research question, forming a strong foundation for the design of the proposed detection technique.

The second objective to develop a new detection method called **DetectCombined** was accomplished through the integration of pattern-based filtering, input validation enhancement, and keyword-matching logic. The DetectCombined model was designed to overcome the limitations of traditional PHP-based validation, which often fails to detect complex or obfuscated SQL payloads. The implementation phase involved embedding layered security rules that work together to detect anomalies in user inputs more efficiently. The results presented in this chapter confirm that the proposed technique achieved superior detection accuracy compared to conventional methods. DetectCombined successfully minimized false negatives, accurately detected mixed attack patterns, and improved the overall reliability of web input validation systems. This achievement directly responds to the second research question, which focused on

whether a new combined detection approach could outperform standard validation techniques.

The third objective to evaluate the effect of DetectCombined on the security of online applications was met through experimental testing and comparative analysis. The findings revealed that web applications utilizing DetectCombined experienced a significant improvement in SQLi detection and prevention performance. Compared with traditional PHP input validation, the proposed method demonstrated higher precision, reduced system vulnerability, and better adaptability to varying types of SQL attack vectors. The evaluation results clearly showed that the technique not only strengthens database protection but also enhances user data integrity and system stability. This outcome directly answered the third research question, confirming that the implementation of DetectCombined has a measurable positive effect on web application security.

Overall, the analysis presented in Chapter 4 validates that all research objectives have been successfully achieved and all corresponding questions adequately addressed. The study established a direct link between identifying SQLi threats, developing a robust detection solution, and assessing its effectiveness in real-world conditions. The collective findings affirm that the DetectCombined technique significantly enhances the protection of user input data, reduces the likelihood of database compromise, and improves overall system resilience against malicious SQL injection attacks. Consequently, the results contribute valuable practical and theoretical insights into modern web application security by providing a validated detection framework that can serve as a reliable defense mechanism for future secure web system designs.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

This chapter discusses the findings presented in the previous section in relation to the study objectives and research questions. It interprets how the results support the effectiveness of the proposed **DetectCombined** technique in addressing modern SQL injection (SQLi) challenges faced by web applications. The discussion connects the experimental outcomes with existing literature on SQLi detection and prevention, highlighting how this study contributes to advancing security mechanisms beyond conventional PHP-based input validation. The improved detection accuracy and reduced false negatives achieved by DetectCombined confirm that combining multiple detection strategies can significantly strengthen web application security. Furthermore, this chapter explores the broader implications of these findings for developers, system administrators, and security researchers, emphasizing the importance of continuous innovation in web security practices to counter the evolving nature of malicious SQL attacks.

5.2 Conclusion

The review of literature and the results of prototype technique (DetectCombined) reveal that the attacks of malicious SQL injection could be blocked if all vulnerable gaps in the validation protocol in input fields have been effectively filtered and encrypted before sending SQL queries to the target databases in the remote server. The various techniques found in the literature shows that SQL injection is one of the highest threats to all kinds of industries, e.g., banks, business, service companies, as well as government agencies. Therefore, the analysis of these techniques would help to enhance the shield against these kinds of attacks and protect the information in the databases of organizations from being damaged or deleted. The results of this study show that the prevention malicious SQL injection should be in the login stage and not after. Therefore,

the technique that has been developed in this study is based on three stages, the first stage is filtration the text of all input fields and not limited to the username and password, the second stage encrypt the data in both sides, client-side and server-side. The third stage is the most important part of DetectCombined which is storing previous SQL attacks into a history table in the remote database to increase the effectiveness and accuracy of input validation.

It is concluded unfiltered inputs in forms is always bears high risk to the data stored in databases associated with input forms. The parametrizes and variables that passed to PHP code in the server to process SQL query are the main tools used by attackers to deliver their malicious query to the target database. Therefore, it is highly important to perfectly filtering the input data in websites through defined input fields to prevent all kinds of malicious text that include damaging SQL queries injected so that delete or alter part or completely the databases in a remote server of vendors. The injected commands that can be used to exploit SQL injection vulnerability and execute some SQL commands that cause huge financial losses for big companies or banks. Thereby, the study highlights the risks of SQL injection attacks on the security and confidentially of data stored in databases. The malicious SQL commands pass in each parameter to the query later. The method used in this study is based on new technique (filtration, validation, and history) that allows the protection code distinguish between safe SQL commands and malicious ones, regardless of the types of users whether a general user of admin.

It is evident, that scholars in this topic emphasized on a fact that weak filtering to the input fields in any webform, especially the input fields for username and password is potentially risky especially if the remote database stores confidential data like credit card information. Many developers neglect adding a history of previous attacks to their techniques so that ease and fasten the process of distinguishing from a real user and a malicious one in a dynamic web application such as e-commerce and web portals of banks. Neglecting these points permits the hacker to execute a malicious code and delete

certain tables belong to the database or deleted the whole database which cause a huge problem for organizations deal with critical and sensitive information.

In sum, the method developed in this study successfully tested a prototype webpage filtering and validation to input fields and for new or previous SQL injection attempts of malicious parameter to the database as text include SQL commands that can be used incorrectly to access the data stored in all rows (records) of tables in the remote database. The webpages used to enter personal data of users and admins could be shielded through DetecCombined code to protect a web application from any compromise to the security of the database in the server side.

5.3 Research contribution

There are two contributions in this study, the first is developing a novel validation and filtering code termed “DetectCombined”, the second contribution is that integration of history of previous malicious SQL attacks with new attacks to enhance the validation process and boost the separation of new kinds of attacks and former ones, this technique will produce a robust protection methods that increase the effectiveness and accuracy of input validation, which in turn increase the satisfaction of vendors of such application and enhance the safety of confidential records especially in sensitive organizations, such as banks, law enforcement agencies, financial institutions, and public service entities. The proposed DetectCombined used in this study is an innovated technique that execute a protection code based on a sequence of three stages: filtration-validation-history, this technique produces a robust protection code that distinguish between safe SQL commands and malicious ones, and reinforce the memory of detection procedure by saving previous SQL attacks in special tables in the remote database, regardless of the types of users whether a general user of admin.

In addition, this study enhances the body of knowledge on SQL attacks by focusing on vulnerability gaps in the filtration protocol before an attacker send malicious SQL queries that cause huge financial losses for big companies or banks. Thereby, the findings of this study are important to increase the awareness of these kinds of attacks on the security and confidentially of people and companies.

5.4 Limitations of Study

The novelty of the proposed DetectCombined technique lies in its integrated three-step approach that addresses SQL injection threats comprehensively: filtering all input fields, encrypting data on both client and server sides, and leveraging a history table to store past SQL injection patterns for enhanced input validation. This multi-layered defence mechanism not only prevents malicious SQL commands during the login process but also strengthens database security across various input fields, which is a significant improvement over conventional methods. The benefits of DetectCombined include proactive prevention of SQL injection attacks, safeguarding sensitive user data, and enhancing the overall integrity of databases for web applications.

In sum, the DetectCombined technique introduces a novel and comprehensive approach to mitigating SQL injection attacks by integrating three critical steps: filtering all input fields, encrypting data on both client and server sides, and using a history table to store and analyze past SQL injection attempts. This layered approach addresses existing gaps in traditional methods that often focus solely on login credentials or reactive detection after an attack occurs. By incorporating these measures, DetectCombined not only prevents malicious SQL commands during the login process but also secures all input fields across the web application, ensuring a more holistic defence. The history table, in particular, adds an innovative element by leveraging past attack patterns to enhance the precision and efficacy of input validation, enabling the system to adapt and improve over time. The benefits of this technique are manifold. It proactively safeguards sensitive user data and prevents unauthorized access to databases, making it especially valuable for industries such as banking, government, and e-commerce, where data integrity and confidentiality are paramount. By implementing encryption at both ends, it reduces the likelihood of data interception during transmission, further fortifying the security framework. Moreover, the technique's ability to block new and previously

recorded SQL injection attempts demonstrates its robustness and adaptability in dynamic threat environments. However, there are potential limitations to consider. The added computational overhead from encryption processes and the history table lookups may impact system performance, particularly for high-traffic web applications or large-scale databases. Additionally, maintaining and updating the history table with new attack patterns requires careful resource allocation to avoid bottlenecks. Despite these challenges, the DetectCombined technique offers a significant advancement in SQL injection prevention by providing a proactive, adaptable, and multi-faceted security solution that enhances the resilience of web applications against one of the most persistent cyber threats.

5.5 Future Studies

The benefits of this technique are manifold. It proactively safeguards sensitive user data and prevents unauthorized access to databases, making it especially valuable for industries such as banking, government, and e-commerce, where data integrity and confidentiality are paramount. By implementing encryption at both ends, it reduces the likelihood of data interception during transmission, further fortifying the security framework. Moreover, the technique's ability to block new and previously recorded SQL injection attempts demonstrates its robustness and adaptability in dynamic threat environments.

However, there are potential limitations to consider. The added computational overhead from encryption processes and the history table lookups may impact system performance, particularly for high-traffic web applications or large-scale databases. Additionally, maintaining and updating the history table with new attack patterns requires careful resource allocation to avoid bottlenecks.

Future studies can further investigate and extend the body of knowledge in this domain by exploring several directions. For instance, the integration of machine learning or artificial intelligence could enable dynamic detection of previously unseen SQL injection patterns, improving adaptability against evolving threats. Comparative analyses of DetectCombined against other contemporary SQL injection prevention

frameworks in large-scale and high-traffic environments could provide deeper insights into scalability and performance optimization.

Furthermore, research could explore automated updates to the history table using real-time threat intelligence feeds, reducing manual maintenance and improving responsiveness to emerging attacks. Additional work may also investigate the applicability of DetectCombined across diverse web technologies and database management systems to generalize its effectiveness in broader contexts.

Despite these challenges, the DetectCombined technique offers a significant advancement in SQL injection prevention by providing a proactive, adaptable, and multi-faceted security solution that enhances the resilience of web applications against one of the most persistent cyber threats.

5.6 Recommendations

Based on the findings and result, the study suggests the following recommendations:

i. Using a history records of previous SQL attacks will enhance the security of webpages the include input fields and linked to a remote database.

ii. Avoid any weakness in SQL server by providing effective input validation-filtering and encryption to discriminate the malicious parameters used for injection SQL attack queries that may damage the whole data in a target database. The most important precautions are data sanitization and validation, which should already be in place.

iii. Weak filtering of user input will enable a hacker to insert SQL malicious code depending on special SQL statement and permitting the hacker to execute a malicious code.

iv. Use multiple detection methods for SQL injection. Today, no single solution is sufficient to defeat SQL injection attacks. This is due to the power of SQL and the flexibility that it gives to the user. The DetecCombined should fill all the previous gaps in this domain.

v. It is necessary to set security guidelines for the operators of databases in any organization rely on confidential information from clients so that protecting sensitive information from being hacked.

vi. It is very important to update and patch the database security by enhancement the vulnerabilities in web applications and databases that hackers could exploit through SQL injection from time to time on regular bases when new injection techniques are discovered, so it's vital to apply patches and updates as soon as practical.

vii. To reduce SQL attacking surface by getting rid of any poor database functionality that may enable the hacker to take advantage of it and start injection his malicious code.

REFERENCES

Abdel-Rahman, M. (2023). Advanced cybersecurity measures in IT service operations and their crucial role in safeguarding enterprise data in a connected world. *Eigenpub Review of Science and Technology*, *7*(1), 138-158.

Abdullayev, V., & Chauhan, A. S. (2023). SQL injection attack: Quick view. *Mesopotamian Journal of CyberSecurity*, *2023*, 30-34.

Abdullayev, V., & Chauhan, A. S. (2023). SQL injection attack: Quick view. *Mesopotamian Journal of CyberSecurity*, *2023*, 30-34.

Accenture. (2019). Global value of risks from direct and indirect cyberattacks, cumulative 2019 to 2023. Retrieved April 16, 2000. https://res.cloudinary.com/yumyoshojin/image/upload/v1/pdf/fighting-fraud-2019.pdf

Acunetix. (2023). *Advanced SQL Injection Techniques.* Retrieved from https://www.acunetix.com/blog/articles/advanced-sql-injection-techniques/

Adebiyi, M. O., Arowolo, M. O., Archibong, G. I., Mshelia, M. D., & Adebiyi, A. A. (2021, December). An Sql Injection Detection Model Using Chi-Square with Classification Techniques. *In 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)* (pp. 1-8). IEEE.

Aliero, M. S., Ghani, I., Qureshi, K. N., & Rohani, M. F. A. (2020). An algorithm for detecting SQL injection vulnerability using black-box testing. *Journal of Ambient Intelligence and Humanized Computing*, 11(1), 249-266.

Aliero, M. S., Ghani, I., Zainudden, S., Khan, M. M., & Bello, M. (2015). Review on SQL injection protection methods and tools. *Jurnal Teknologi*, *77*(13), 49-66.

Al-Khashab, E., Al-Anzi, F. S., & Salman, A. A. (2011, April). PSIAQOP: preventing SQL injection attacks based on query optimization process. In *Proceedings of the Second Kuwait Conference on e-Services and E-Systems* (pp. 1-8).

Al-Maliki, M. H., & 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.

Al-Maliki, M. H., & 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.

Anley, C. (2002). Advanced SQL Injection in SQL Server Applications. NGSSoftware Insight Security Research.

Appelt, D., Nguyen, C. D., Briand, L. C., & Alshahwan, N. (2014, July). Automated testing for SQL injection vulnerabilities: an input mutation approach. In Proceedings of the 2014 International Symposium on Software Testing and Analysis (pp. 259-269).

Aung, T. M., & Hla, N. N. (2022). Recent Techniques for Exploitation and Protection of Common Malicious Inputs to Online Applications. *Cyber Security and Digital Forensics*, 335-360.

Avireddy, S., Perumal, V., Gowraj, N., Kannan, R. S., Thinakaran, P., Ganapthi, S., & Prabhu, S. (2012, June). Random4: An application specific randomized encryption algorithm to prevent SQL injection. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications* (pp. 1327-1333). IEEE.

Avireddy, S., Perumal, V., Gowraj, N., Kannan, R. S., Thinakaran, P., Ganapthi, S., & Prabhu, S. (2012, June). Random4: An application specific randomized encryption algorithm to prevent SQL injection. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications* (pp. 1327-1333). IEEE.

Avireddy, S., Perumal, V., Gowraj, N., Kannan, R. S., Thinakaran, P., Ganapthi, S., & Prabhu, S. (2012, June). Random4: An application specific randomized encryption algorithm to prevent SQL injection. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications* (pp. 1327-1333). IEEE.

Azman, M., Marhusin, M. F., & Sulaiman, R. (2021). Machine Learning-Based Technique to Detect SQL Injection Attack. *Journal of Computer Science*, 17. 296-303.

Balapour, A., Nikkhah, H. R., & Sabherwal, R. (2020). Mobile application security: Role of perceived privacy as the predictor of security perceptions. *International Journal of Information Management*, *52*, 102063.

Bayyapu, N. (2021). SQL Injection Attacks and Mitigation Strategies: The Latest Comprehension. *In Advances in Cybersecurity Management* (pp. 199-220). Springer, Cham.

BBC News. (2018). British Airways fined £183m over data breach. Retrieved from https://www.bbc.com

Betarte, G., Pardo, Á., & Martínez, R. (2018, December). Web application attacks detection using machine learning techniques. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 1065-1072). IEEE.

Bisht, P., Madhusudan, P., & Venkatakrishnan, V. N. (2010). CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. *ACM Transactions on Information and System Security (TISSEC)*, *13*(2), 1-39.

Brightsec. (2023). SQL Injection Attack: How It Works, Examples and Prevention. Retrieved from https://brightsec.com/sql-injection-attack

Busch, M. (2016). *Evaluating & engineering: an approach for the development of secure web applications* (Doctoral dissertation, Dissertation, München, Ludwig-Maximilians-Universität, 2016).

Busch, M., & Wirsing, M. (2015). An Ontology for Secure Web Applications. *Int. J. Softw. Informatics*, *9*(2), 233-258.

Check Point Software Technologies. (2023). Preventing SQL Injection Attacks: Best Practices. Check Point. Retrieved from Check Point SQLi Prevention.

Chen, D., Yan, Q., Wu, C., & Zhao, J. (2021). Sql injection attack detection and prevention techniques using deep learning. *In Journal of Physics: Conference Series* (Vol. 1757, No. 1, p. 012055). IOP Publishing.

Chen, P., Wu, S., & Chiang, M. (2019). AI-driven Dynamic Patch Management for Web Applications. *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 1287-1298.

Chen, T., Weng, Z., Chen, Y., Jin, C., Lv, M., Zhu, T., & Lin, J. (2021). WebSmell: An Efficient Malicious HTTP Traffic Detection Framework Using Data Augmentation. In *Information Security and Cryptology: 16th International Conference, Inscrypt 2020, Guangzhou, China, December 11–14, 2020, Revised Selected Papers* (pp. 193-201). Springer International Publishing.

Chess, B., & McGraw, G. (2004). Static Analysis for Security. IEEE Security & Privacy, 2(6), 76-79.

Choo, K. K. R., Liu, L., & Liu, Y. (2019). Towards automating web application vulnerability discovery using machine learning. *Future Generation Computer Systems*, 98, 365-379.

Churi, P. P., & Mistry, K. (2017). DE-PRE-Tool for Detection and Prevention from Input Validation Attacks on Website. *Circulation in Computer Science*, 2(5), 23-27.

Ciampa, A., Visaggio, C. A., & Penta, M. D. (2010). A heuristic-based approach for detecting SQL-injection. *Proceedings of the 2010 ICSE*.

Clarke, J. (2009). SQL Injection Attacks and Defence. Syngress.

Clarke, J. (2009). SQL Injection Attacks and Defence. Syngress.

Crişan, A., Florea, G., Halasz, L., Lemnaru, C., & Oprisa, C. (2020, September). Detecting malicious URLs based on machine learning algorithms and word embeddings. In *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)* (pp. 187-193). IEEE.

CWE/SANS. (2011). CWE/SANS Top 25 Most Dangerous Software Errors. Retrieved from https://cwe.mitre.org/top25/

Das, D., Sharma, U., & Bhattacharyya, D. K. (2019). Defeating SQL injection attack in authentication security: an experimental study. International Journal of Information Security, 18, 1-22.

Das, P., & Datta, S. (2022). SQL Injection Prevention System Using PHP. AIJR Abstracts, 85.

Dasmohapatra, S., & Priyadarshini, S. B. B. (2022). A Comprehensive Study on SQL Injection Attacks, Their Mode, Detection and Prevention. *In Proceedings of*

Second Doctoral Symposium on Computational Intelligence (pp. 617-632). Springer, Singapore.

Deepa, G., Thilagam, P. S., Khan, F. A., Praseed, A., Pais, A. R., & Palsetia, N. (2018). Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications. *International Journal of Information Security*, *17*, 105-120.

Deriba, F. G., Kassa, T. M., & Demilie, W. B. (2022). Attacks on SQL Injection and Developing Compressive Framework Using a Hybrid and Machine Learning Approach.

Dhanare, R., Sharma, P. C., & Srivastava, D. K. (2021, December). Vulnerabilities, Attacks and Solutions of Cybersecurity in Medical Domain. *In 2021 International Conference on Computational Performance Evaluation (ComPE)* (pp. 034-039). IEEE.

D'silva, K., Vanajakshi, J., Manjunath, K. N., & Prabhu, S. (2017, May). An effective method for preventing SQL injection attack and session hijacking. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)* (pp. 697-701). IEEE.

DZone. (2023). OWASP Top 10 Explained: SQL Injection. Retrieved from https://dzone.com/articles/owasp-top-10-explained-sql-injection

Ebrahimi, M. (2021). *AI-Enabled Cybersecurity Analytics: Detecting and Defending Against Cyber Threats*. University of Arizona. Retrieved from University of Arizona Repository.

Fang, Y., Peng, J., Liu, L., & Huang, C. (2018, March). WOVSQLI: Detection of SQL injection behaviors using word vector and LSTM. In *Proceedings of the 2nd international conference on cryptography, security and privacy* (pp. 170-174).

Fazzini, K. (2019). Equifax to pay up to $700 million in data breach settlement. CNBC. Retrieved from https://www.cnbc.com

FireEye. (2023). *Detecting and Preventing SQL Injection Attacks.* Retrieved from https://www.fireeye.com/blog/threat-research/2023/detecting-preventing-sql-injection-attacks.html

Fu, H., Guo, C., Jiang, C., Ping, Y., & Lv, X. (2023). SDSIOT: An SQL Injection Attack Detection and Stage Identification Method Based on Outbound Traffic. *Electronics*, *12*(11), 2472.

George, T. K., Jacob, K. P., & James, R. K. (2018). Token based detection and neural network-based reconstruction framework against code injection vulnerabilities. *Journal of Information Security and Applications*, *41*, 75-91.

Ghafari, M. (2017). SQL Injection Attacks and Countermeasures. In *Proceedings of the International Conference on Information Security*.

Gogoi, B., Ahmed, T., & Dutta, A. (2021, December). Defending against SQL Injection Attacks in Web Applications using Machine Learning and Natural Language Processing. *In 2021 IEEE 18th India Council International Conference (INDICON)* (pp. 1-6). IEEE.

Goodin, D. (2012). Zappos data breach hits 24 million customers. Ars Technica. Retrieved from https://arstechnica.com

Goodin, D. (2019). Cisco patches critical SQL injection flaw in Prime License Manager software. Ars Technica. Retrieved from https://arstechnica.com

Goyal, A., & Matta, P. (2023, September). Beyond the Basics: A Study of Advanced Techniques for Detecting and Preventing SQL Injection Attacks. In *2023 4th International Conference on Smart Electronics and Communication (ICOSEC)* (pp. 628-631). IEEE.

Greenberg, A. (2012). Zappos Breach Exposes Millions of Customer Records. Forbes. Retrieved from https://www.forbes.com

Gupta, A., & Sharma, L. S. (2022). A Novel Approach for Detecting SQL Injection Attacks Using Snort. *Journal of The Institution of Engineers (India): Series B*, 1-9.

HackerOne. (2024). SQL Injection Attack: How It Works and 4 Preventive Measures. Retrieved from https://www.hackerone.com/blog/sql-injection-attack-how-it-works-and-4-preventive-measures

HackTheBox. (2024). SQL Injection in the Wild. Retrieved from https://www.hackthebox.com/blog/sql-injection-in-the-wild

Hadabi, A., Elsamani, E., Abdallah, A., & Elhabob, R. (2022). An Efficient Model to Detect and Prevent SQL Injection Attack. *Journal of Karary University for Engineering and Science.*

Halder, R., & Cortesi, A. (2010, June). Obfuscation-based analysis of SQL injection attacks. In *The IEEE symposium on Computers and Communications* (pp. 931-938). IEEE.

Halfond, W. G. J., Viegas, J., & Orso, A. (2006). A Classification of SQL-Injection Attacks and Countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*.

Halfond, W. G. J., Viegas, J., & Orso, A. (2006). A Classification of SQL-Injection Attacks and Countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*.

Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. *Proceedings of the IEEE International Symposium on Secure Software Engineering*, 13-15.

Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. Proceedings of the IEEE International Symposium on Secure Software Engineering, 13-15.

Hassan, W., Nazir, B., & Khan, S. U. (2017). Anomaly detection in SQL injection attacks: A machine learning approach. *Computers & Security*, 68, 223-241.

Hlaing Z. C., Khaing M. (2020). A Detection and Prevention Technique on SQL Injection Attacks. *IEEE Conference on Computer Applications(ICCA*), 1-6.

Howard, M., & LeBlanc, D. (2003). Writing Secure Code (2nd ed.). Microsoft Press.

Howard, M., & LeBlanc, D. (2003). Writing Secure Code (2nd ed.). Microsoft Press.

Huang, D., Allen, T. T., Notz, W. I., & Zeng, N. (2006). Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of global optimization*, *34*(3), 441-466.

IBM Security. (2022). Cost of a Data Breach Report. IBM. Retrieved from IBM Cost of a Data Breach

Imperva. (2023). *SQL Injection Attack.* Retrieved from https://www.imperva.com/learn/application-security/sql-injection-sqli/

Ines, J., Omar, C., Habib, H., & Adel, M. (2020). SQL Injection Attack Detection and Prevention Techniques Using Machine Learning. *International Journal of Applied Engineering Research*, 15(6), 569-580.

Ingre, B., Yadav, A., & Soni, A. K. (2018). Decision tree-based intrusion detection system for NSL-KDD dataset. In *Information and Communication Technology for Intelligent Systems (ICTIS 2017)-Volume 2 2* (pp. 207-218). Springer International Publishing.

Jacob, I., & Pirnau, M. (2020). SQL INJECTION ATTACKS AND VULNERABILITIES. *Journal of Information Systems & Operations Management*, 68-81.

Jagadeesan, M., Selvaraj, P. A., & Sanchana, V. (2020). SQL Injection Attack Discovery and Defending Mechanism for Multi-Tier Web Applications. *International Journal of Applied Engineering Research*, 15(4), 405-408.

Jamal, H., Algeelani, N. A., & Al-Sammarraie, N. (2024). Safeguarding data privacy: strategies to counteract internal and external hacking threats. *Computer Science and Information Technologies*, *5*(1), 46-54.

Jana, A., & Maity, D. (2020, July). Code-based analysis approach to detect and prevent SQL injection attacks. *In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)* (pp. 1-6). IEEE.

Janeja, V. P. (2022). *Data Analytics for Cybersecurity*. Cambridge University Press.

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.

Joshi, A., & Geetha, V. (2014, July). SQL Injection detection using machine learning. In *2014 international conference on control, instrumentation, communication and computational technologies (ICCICCT)* (pp. 1111-1115). IEEE.

Juan, C. Urko, Z., & Ricardo J. R. (2016). Intrusions and Malware, and Vulnerability Assessment. Springer, 2016, 37-57.

Kar, D., Panigrahi, S., & Sundararajan, S. (2016). SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM. *Computers & Security*, 60, 206-225.

Kareem, F. Q., Ameen, S. Y., Salih, A. A., Ahmed, D. M., Kak, S. F., Yasin, H. M., ... & Omar, N. (2021). SQL injection attacks prevention system technology. *Asian Journal of Research in Computer Science*, 13, 32.

Kasim, Ö. (2021). An ensemble classification-based approach to detect attack level of SQL injections. *Journal of Information Security and Applications*, *59*, 102852.

Kasowaki, L., & Ali, K. (2024). *Cyber Hygiene: Safeguarding Your Data in a Connected World* (No. 11698). EasyChair.

Katole, R. A., Sherekar, S. S., & Thakare, V. M. (2018, January). Detection of SQL injection attacks by removing the parameter values of SQL query. *In 2018 2nd International Conference on Inventive Systems and Control (ICISC)* (pp. 736-741). IEEE.

Kaur, K., & Kaur, R. (2019). Role of web application firewalls in securing web applications against SQL injection attacks. International Journal of Computer Applications, 182(42), 34-38.

Khan, J. R., Farooqui, S. A., & Siddiqui, A. A. (2023). A Survey on SQL Injection Attacks Types & their Prevention Techniques. *Journal of Independent Studies and Research Computing*, *21*(2), 1-4.

Kim, H., & Zeldovich, N. (2018). SQLrand: Preventing SQL injection attacks using reinforcement learning. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 620-633.

Kim, H., & Zeldovich, N. (2018). SQLrand: Preventing SQL injection attacks using reinforcement learning. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 620-633.

Kolias, C., Kambourakis, G., Stavrou, A., & Gritzalis, S. (2016). Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset. *Computer Networks*, 109, 144-160.

Kranthikumar, B., & Velusamy, R. L. (2020). SQL injection detection using REGEX classifier. *Journal of Xi'an University of Architecture & Technology*, 12(6), 800-809.

Krebs, B. (2009). Heartland Payment Systems: Breach Exposes Up to 100 Million Cards. Retrieved from https://krebsonsecurity.com

Kumar, A., Dutta, S., & Pranav, P. (2024). Analysis of SQL injection attacks in the cloud and in WEB applications. *Security and Privacy*, *7*(3), e370.

Kumar, D. G., & Chatterjee, M. (2014). Detection block model for SQL injection attacks. *International Journal of Computer Network and Information Security*, *6*(11), 56-63.

Kumar, D. G., & Chatterjee, M. (2015). MAC based solution for SQL injection. *Journal of Computer Virology and Hacking Techniques*, *11*, 1-7.

Kuroki, K., Kanemoto, Y., Aoki, K., Noguchi, Y., & Nishigaki, M. (2020, July). Attack Intention Estimation Based on Syntax Analysis and Dynamic Analysis for SQL Injection. *In 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 1510-1515). IEEE.

Laughter, A., Omari, S., Szczurek, P., & Perry, J. (2021). Detection of malicious http requests using header and url features. In *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 2* (pp. 449-468). Springer International Publishing.

Leyden, J. (2016). TalkTalk fined £400,000 for security failings over data breach. The Register. Retrieved from https://www.theregister.com

Li, Q., Wang, F., Wang, J., & Li, W. (2019). LSTM-based SQL injection detection method for intelligent transportation system. *IEEE Transactions on Vehicular Technology*, *68*(5), 4182-4191.

Lindstrom, P., Myrbakken, H., & Jaatun, M. G. (2019). The use of parameterized queries to prevent SQL injection attacks in web applications. International Journal of Computer Science and Information Security, 17(4), 101-110.

Lu, D., Fei, J., & Liu, L. (2023). A semantic learning-based SQL injection attack detection technology. *Electronics*, *12*(6), 1344.

Lu, D., Fei, J., & Liu, L. (2023). A semantic learning-based SQL injection attack detection technology. *Electronics*, *12*(6), 1344.

Madhusudhan, R., & Ahsan, M. (2022). Prevention of SQL Injection Attacks Using Cryptography and Pattern Matching. *In International Conference on Advanced Information Networking and Applications* (pp. 624-634). Springer, Cham.

Makiou, A., Begriche, Y., & Serhrouchni, A. (2014, November). Improving Web Application Firewalls to detect advanced SQL injection attacks. In *2014 10th international conference on information assurance and security* (pp. 35-40). IEEE.

Mamadhan, S., Manesh, T., & Paul, V. (2012, November). SQLStor: Blockage of stored procedure SQL injection attack using dynamic query structure validation. In *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)* (pp. 240-245). IEEE.

Manjunatha, K. M., & Kempanna, M. (2019). RAMIFICATION ANALYSIS OF SQL INJECTION DETECTION IN WEB APPLICATION. *International Journal of Computer Science and Information Security (IJCSIS),* 17(7).

Manoj, R. J., Chandrasekhar A., & Praveena, M. A. (2014). An approach to detect and prevent tautology type SQL injection in web service based on schema validation. *International Journal of Engineering And Computer Science*, 2319-7242.

Marashdeh, Z., Suwais, K., & Alia, M. (2021, July). A survey on sql injection attack: Detection and challenges. In *2021 International Conference on Information Technology (ICIT)* (pp. 957-962). IEEE.

Masango, M., Mouton, F., Antony, P., & Mangoale, B. (2017, September). Web defacement and intrusion monitoring tool: Wdimt. In *2017 International Conference on Cyberworlds (CW)* (pp. 72-79). IEEE.

McAfee. (2023). *SQL Injection: Understanding the Threat.* Retrieved from https://www.mcafee.com/enterprise/en-us/security-awareness/sql-injection.html

McGraw, G. (2006). Software Security: Building Security In. Addison-Wesley.

McGraw, G. (2006). Software Security: Building Security In. Addison-Wesley.

McMillan, R. (2011). Sony Pictures Hacker Sentenced to a Year in Prison. The Wall Street Journal. Retrieved from https://www.wsj.com

Medeiros, I., Beatriz, M., Neves, N., & Correia, M. (2019). SEPTIC: detecting injection attacks and vulnerabilities inside the DBMS. *IEEE Transactions on Reliability*, *68*(3), 1168-1188.

Mehta, P., Sharda, J., & Das, M. L. (2015). SQLshield: preventing SQL injection attacks by modifying user input data. In *Information Systems Security: 11th International Conference, ICISS 2015, Kolkata, India, December 16-20, 2015. Proceedings 11* (pp. 192-206). Springer International Publishing.

Mirza, N., Elhoseny, M., Umar, M., & Metawa, N. (2023). Safeguarding FinTech innovations with machine learning: Comparative assessment of various approaches. *Research in International Business and Finance*, *66*, 102009.

Mitchell, C. (2005). Security Risk Management. Wiley.

Mitchell, C. (2005). Security Risk Management. Wiley.

Monovm. (2024). Best SQL Injection (SQLi) Detection Tools for 2024. Retrieved from Monovm

Monovm. (2024). Best SQL Injection (SQLi) Detection Tools for 2024. Retrieved from Monovm

Moosa, A. (2019). Artificial neural network-based web application firewall for SQL injection. *World Academy of Science, Engineering and Technology*, 4, 2010.

Nadeem, R. M., Saleem, R. M., Bashir, R., & Habib, S. (2017). Detection and prevention of SQL injection attack by dynamic analyzer and testing model. *International Journal of Advanced Computer Science and Applications*, *8*(8), 209-214.

Nasereddin, M., ALKhamaiseh, A., Qasaimeh, M., & Al-Qassas, R. (2021). A systematic review of detection and prevention techniques of SQL injection attacks. *Information Security Journal: A Global Perspective*, 1-14.

Nasereddin, M., ALKhamaiseh, A., Qasaimeh, M., & Al-Qassas, R. (2023). A systematic review of detection and prevention techniques of SQL injection attacks. *Information Security Journal: A Global Perspective*, *32*(4), 252-265.

Nature. (2023). *Investigating the possibility of data leakage in time of live VM migration*. Nature. Retrieved from Nature.

Nithya, V., Regan, R., & Vijayaraghavan, J. (2013). A survey on SQL injection attacks, their detection and prevention techniques. *International Journal of Engineering and Computer Science*, *2*(4), 886-905.

NVD. (2019). CVE-2019-1821 Detail. Retrieved from https://nvd.nist.gov

Oudah, M. A., & Marhusin, M. F. (2024). SQL Injection Detection using Machine Learning: A Review. *Malaysian Journal of Science Health & Technology*, *10*(1), 39-49.

OWASP, Owasp top ten project, https://www.owasp.org/index.php/Category:OWASP Top Ten Project, 2019, accessed on March 2023

OWASP. (2017). OWASP top ten. Open Web Application Security Project. Retrieved from https://owasp.org/www-project-top-ten/

OWASP. (2021). OWASP Top Ten. Retrieved from https://owasp.org/www-project-top-ten/

OWASP. (2021). OWASP Top Ten. Retrieved from https://owasp.org/www-project-top-ten/

OWASP. (2022). Input validation. Open Web Application Security Project. Retrieved from https://owasp.org/www-community/Input_Validation

OWASP. (2023). *SQL Injection.* Retrieved from https://owasp.org/www-community/attacks/SQL_Injection

OWASP. (2024). SQL Injection Prevention Cheat Sheet. Retrieved from OWASP

OWASP. (2024). SQL Injection Prevention Cheat Sheet. Retrieved from OWASP

Pagliery, J. (2014). Heartland: Lessons from a Huge Data Breach. CNNMoney. Retrieved from https://money.cnn.com

Panigrahi, R., Borah, S., Pramanik, M., Bhoi, A. K., Barsocchi, P., Nayak, S. R., & Alnumay, W. (2022). Intrusion detection in cyber–physical environment using hybrid Naïve Bayes-Decision table and multi-objective evolutionary feature selection. *Computer Communications*, 188, 133-144.

Parashar, D., Sanagavarapu, L. M., & Reddy, Y. R. (2021, February). SQL Injection Vulnerability Identification from Text. *In 14th Innovations in Software Engineering Conference (formerly known as India Software Engineering Conference)* (pp. 1-5).

Pentest-Tools.com. (2024). Breaking Down the 5 Most Common SQL Injection Attacks. Retrieved from https://pentest-tools.com/blog/sql-injection

Perkins, J., Eikenberry, J., Coglio, A., Willenson, D., Sidiroglou-Douskos, S., & Rinard, M. (2016). AutoRand: Automatic keyword randomization to prevent injection

attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13* (pp. 37-57). Springer International Publishing.

Ponemon Institute. (2023). Consumer Sentiment Report: Data Breach Impacts. Ponemon Institute. Retrieved from Ponemon Institute Reports

PortSwigger. (2024). Latest SQL Injection Security News. The Daily Swig. Retrieved from https://portswigger.net/daily-swig/sql-injection

Poulsen, K. (2006). MySpace Worm Takes Over Million Accounts. Wired. Retrieved from https://www.wired.com

Praveen Kumar. (2013). The Multi-Tier Architecture for Developing Secure Website with Detection and Prevention of SQL-Injection Attacks. *International Journal of Computer Applications*, 62(9), 30-36.

Qu, Z., Ling, X., Wang, T., Chen, X., Ji, S., & Wu, C. (2024). AdvSQLi: Generating Adversarial SQL Injections against Real-world WAF-as-a-service. *IEEE Transactions on Information Forensics and Security*.

Qualys. (2024). Guarding against SQL injection: Techniques to enhance code security. Retrieved from https://www.qualys.com/blog

Rai, S., & Nagpal, B. (2019). Detection & Prevention of SQL Injection Attacks: Developments of the Decade.

Raniah, A. (2019). SQL Injection Attacks: Detection And Prevention Techniques. 3rd International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), Oct 8-10, 2014, AIIT, Amity University Uttar Pradesh, Noida, India

Rapid7. (2023). *SQL Injection in API Endpoints: A Growing Concern.* Retrieved from https://www.rapid7.com/blog/sql-injection-api-endpoints-growing-concern/

Robinson, D. (2019). British Airways fined £183m for data breach affecting 500,000 customers. The Guardian. Retrieved from https://www.theguardian.com

Roobini, M. S., Srividhya, S. R., Vennela, K., & Nikhila, G. (2022, March). Detection of SQL Injection Attack Using Adaptive Deep Forest. *In 2022 International Conference on Communication, Computing and Internet of Things (IC3IoT)* (pp. 1-6). IEEE.

Sadeghian, A., Zamani, M., & Idris, N. B. (2013). SQL injection: Detection and prevention techniques. *International Journal of Security and Its Applications*, 7(1), 23-42.

Safety Detectives. (2024). What Is an SQL Injection Attack? And How to Prevent It in 2024. Retrieved from https://www.safetydetectives.com/sql-injection-attack

Schwartz, M. J. (2011). Sony Pictures Hacker Arrested, Breach Not Over. Retrieved from https://www.databreachtoday.com

Shahriar, H., & Zulkernine, M. (2012). Mitigation of program security vulnerabilities: Approaches and challenges. ACM Computing Surveys, 44(3), 11.

Shahriar, H., & Zulkernine, M. (2012, October). Information-theoretic detection of SQL injection attacks. In *2012 IEEE 14th international symposium on high-assurance systems engineering* (pp. 40-47). IEEE.

Sheykhkanloo, N. M. (2017). A learning-based neural network model for the detection and classification of SQL injection attacks. *International Journal of Cyber Warfare and Terrorism (IJCWT)*, *7*(2), 16-41.

Shulman, A., & Natan, Y. (2009). SQL Injection Signatures Evasion. In *Proceedings of the Black Hat USA Conference*.

Shulman, A., & Natan, Y. (2009). SQL Injection Signatures Evasion. In *Proceedings of the Black Hat USA Conference*.

Singh, H., & Dua, M. (2018, July). Website attacks: challenges and preventive methodologies. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)* (pp. 381-387). IEEE.

Singh, J. P. (2016). Analysis of SQL injection detection techniques. *arXiv preprint arXiv:1605.02796*.

Sinha, K., & Verma, M. (2021). The Detection of SQL Injection on Blockchain-Based Database. In *Revolutionary Applications of Blockchain-Enabled Privacy and Access Control* (pp. 234-262). IGI Global.

SpringerLink. (2018). *Cyber Threat Intelligence: Challenges and Opportunities*. Springer. Retrieved from SpringerLink.

SQLShack. (2024). SQL Injection: Detection and prevention. Retrieved from SQLShack

SQLShack. (2024). SQL Injection: Detection and prevention. Retrieved from SQLShack

Statista. (2019) Number of web attacks blocked daily worldwide 2015-2018. [Online]. Available: https://www.statista.com/statistics/494961/web-attacksblocked-per-day-worldwide/

Steele, J. (2018). Equifax's Massive 2017 Data Breach Keeps Getting Worse. Wired. Retrieved from https://www.wired.com

Stuttard, D., & Pinto, M. (2011). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws (2nd ed.). Wiley.

Stuttard, D., & Pinto, M. (2011). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws (2nd ed.). Wiley.

Stuttard, D., & Pinto, M. (2011). The web application hacker's handbook: Finding and exploiting security flaws. John Wiley & Sons.

Su, Z. (2007). Advanced techniques for SQL injection attack detection and prevention. IEEE Security & Privacy, 5(3), 26-32.

Su, Z., & Wassermann, G. (2006). The Essence of Command Injection Attacks in Web Applications. In *Proceedings of the 33rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*.

Suto, I. (2008). Analyzing the Accuracy and Time Costs of Static Analysis Tools. In *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation*.

Symantec. (2023). The Aftermath of Data Breaches: Reputation and Recovery. Symantec Corporation. Retrieved from Symantec Breach Aftermath

Symantec. (2023). *The Evolution of SQL Injection Attacks.* Retrieved from https://www.symantec.com/blogs/threat-intelligence/evolution-sql-injection-attacks

Tafa, I., & Resulaj, E. (2021, December). SQL Injection Attacks: Its Types and Ways to Prevent Them. In *BOOK OF PROCEEDINGS* (p. 102).

Thielman, S. (2015). TalkTalk cyber-attack: 156,959 people's data accessed, company says. The Guardian. Retrieved from https://www.theguardian.com

Thomé, J., Shar, L. K., Bianculli, D., & Briand, L. C. (2017, August). Joanaudit: A tool for auditing common injection vulnerabilities. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (pp. 1004-1008).

Thoutam, V. (2022). SQL Injection Vulnerabilities Prevention through ML IPAAS Architecture. *International Journal of Novel Research and Development*, 7(3).

Veeramachaneni, K., Bassias, L., Vo, A., Cornelius, C., & Li, M. (2014). AI2: Training a big data machine to defend. *Proceedings of the 2014 IEEE Security and Privacy Workshops*, 379-383.

Veracode. (2023). *Securing Non-Traditional Databases Against SQL Injection.* Retrieved from https://www.veracode.com/blog/research/securing-non-traditional-databases-against-sql-injection.

Verbruggen, R., & Heskes, T. (2014). Creating firewall rules with machine learning techniques. *Kerckhoffs Institute Nijmegen: Nijmegen, The Netherlands*, 9-27.

Verizon. (2023). 2023 Data Breach Investigations Report. Verizon Enterprise. Retrieved from Verizon DBIR 2023

Wang, Z. (2015). The applications of deep learning on traffic identification. *BlackHat USA*, *24*(11), 1-10.

Wu, T. Y., Chen, C. M., Sun, X., Liu, S., & Lin, J. C. W. (2017). A countermeasure to SQL injection attack for cloud environment. *Wireless Personal Communications*, *96*, 5279-5293.

Xia, L., Semirumi, D. T., & Rezaei, R. (2023). A thorough examination of smart city applications: Exploring challenges and solutions throughout the life cycle with emphasis on safeguarding citizen privacy. *Sustainable Cities and Society*, *98*, 104771.

Xiao, Z., Zhou, Z., Yang, W., & Deng, C. (2017). An approach for SQL injection detection based on behavior and response analysis. *IEEE International Conference on Communication Software and Networks*.

Xiao, Z., Zhou, Z., Yang, W., & Deng, C. (2017). An approach for SQL injection detection based on behavior and response analysis. *IEEE International Conference on Communication Software and Networks*.

Xu, G., Cao, Y., Ren, Y., Li, X., & Feng, Z. (2017). Network security situation awareness based on semantic ontology and user-defined rules for Internet of Things. *IEEE Access*, *5*, 21046-21056.

Yadav, A. K., & Kumar, A. (2022). String Matching Algorithm Based Filter for Preventing SQL Injection and XSS Attacks. *In Inventive Computation and Information Technologies* (pp. 793-807). Springer, Singapore.

Yazeed, A. (2021). An Improved SQL Injection Attack Detection Model Using Machine Learning Techniques. *UTM international journal of innovative computing*, 11(1).

Yu, Y., Yan, H., Ma, Y., Zhou, H., & Guan, H. (2020). DeepHTTP: anomalous HTTP traffic detection and malicious pattern mining based on deep learning. In *Cyber Security: 17th China Annual Conference, CNCERT 2020, Beijing, China, August 12, 2020, Revised Selected Papers 17* (pp. 141-161). Springer Singapore.

Yuan, X., Xue, Y., Li, L., & Liu, H. (2020). An intrusion detection model for SQL injection attacks based on machine learning. *Journal of Ambient Intelligence and Humanized Computing*, 11, 1629-1640.

Yuan, Y., Lu, Y., Zhu, K., Huang, H., Yu, L., & Zhao, J. (2023). A Static Detection Method for SQL Injection Vulnerability Based on Program Transformation. *Applied Sciences*, *13*(21), 11763.

Zetter, K. (2006). MySpace flaw allows mass profile deletion. CNET. Retrieved from https://www.cnet.com

Zhu, Z., Jia, S., Li, J., Qin, S., & Guo, H. (2021, July). SQL Injection Attack Detection Framework Based on HTTP Traffic. In *Proceedings of the ACM Turing Award Celebration Conference-China* (pp. 179-185).

Muhammad, T., & Ghafory, H. (2022). Sql injection attack detection using machine learning algorithm. *Mesopotamian journal of cybersecurity*, *2022*, 5-17.

Alghawazi, M., Alghazzawi, D., & Alarifi, S. (2022). Detection of sql injection attack using machine learning techniques: a systematic literature review. *Journal of Cybersecurity and Privacy*, *2*(4), 764-777.

Kaluža, M., Kalanj, M., & Vukelić, B. (2019). A comparison of back-end frameworks for web application development. *Zbornik veleučilišta u rijeci*, *7*(1), 317-332.

Nair, S. S. (2024). Securing Against Advanced Cyber Threats: A Comprehensive Guide to Phishing, XSS, and SQL Injection Defence. *Journal of Computer Science and Technology Studies*, *6*(1), 76-93.

Zeller, M., & Brehm, L. (2023). *Practical Approaches to Securing Modern Web Applications: A Survey of Common Vulnerabilities and Mitigations.* International Journal of Web Security, 7(2), 45-62.

Venkatramulu, S., Waseem, M. S., Taneem, A., Thoutam, S. Y., & Apuri, S. (2024). Research on SQL injection attacks using word embedding techniques and machine learning. *Journal of Sensors, IoT & Health Sciences*, *2*(01), 55-66.

Kashyap, A., & Jana, A. (2025). A survey: on detection and prevention techniques of SQL injection attacks. *International Journal of Information and Computer Security*, *26*(4), 332-371.

Kumar, A., Dutta, S., & Pranav, P. (2024). Analysis of SQL injection attacks in the cloud and in WEB applications. *Security and Privacy*, *7*(3), e370.

Kashyap, A., & Jana, A. (2025). A survey: on detection and prevention techniques of SQL injection attacks. *International Journal of Information and Computer Security*, *26*(4), 332-371.

Abdullah, H. S., & Abdulazeez, A. M. (2024). Detection of SQL injection attacks based on supervised machine learning algorithms: A review. *International Journal of Informatics, Information System and Computer Engineering (INJIISCOM)*, *5*(2), 152-165.

Arasteh, B., Aghaei, B., Farzad, B., Arasteh, K., Kiani, F., & Torkamanian-Afshar, M. (2024). Detecting SQL injection attacks by binary gray wolf optimizer and machine learning algorithms. *Neural Computing and Applications*, *36*(12), 6771-6792.

Liu, Y., & Dai, Y. (2024). Deep Learning in Cybersecurity: A Hybrid BERT–LSTM Network for SQL Injection Attack Detection. *IET Information Security*, *2024*(1), 5565950.

Fadlil, A., Riadi, I., & Mu'Min, M. A. (2024). Mitigation from SQL Injection Attacks on Web Server using Open Web Application Security Project Framework. *International Journal of Engineering*, *37*(4), 635-645.

Bakır, R. (2025). UniEmbed: A Novel Approach to Detect XSS and SQL Injection Attacks Leveraging Multiple Feature Fusion with Machine Learning Techniques. *Arabian Journal for Science and Engineering*, 1-14.

Aburashed, L., Amoush, M. A., & Alrefai, W. (2024). SQL injection attack detection using machine learning algorithms. *Semarak International Journal of Machine Learning*, *2*(1), 1-12.

Shekhar, K., Sushma, V., Umesh, B., Deepika, K., & Ashish, Y. (2025). Hybrid Text Mining and Generative-AI Framework for SQL Injection Attack Detection. *International Journal of Communication Networks and Information Security*, *17*(3), 744-755.

Ajasa, A. D., Chizari, H., & Alam, A. (2025). Database Security and Performance: A Case of SQL Injection Attacks Using Docker-Based Virtualisation and Its Effect on Performance. *Future Internet*, *17*(4), 156.

Alhowiti, A. H., & Mohamed, A. M. A. (2025, April). A New Database Integrity Protection Approach Against SQL Injection Attacks (SQLIAs). In *2025 4th International Conference on Computing and Information Technology (ICCIT)* (pp. 507-512). IEEE.

Begum, Z., Sravani, P., Priya, M. S., Kumari, P. N., & Rakesh, G. (2025). Reinforcing Web Application Security: A Modified Scheme against SQL Injection Attacks. *Journal of Computational Analysis & Applications*, *34*(4).

Chen, Y., Liang, G., & Wang, Q. (2025). Research on SQL Injection Detection Technology Based on Content Matching and Deep Learning. *Computers, Materials & Continua*, *84*(1).

APPENDIX I

SEARCHING AND DETECTION CODES

```
<?php
session_start();
include("authorization_1.php");

//set session variables in a $_SESSION global array variable
$_SESSION[ 'UserName' ] = 'UserName';
//$_SESSION[ 'role' ] = 'admin';
//$pid=$_SESSION[ 'pid' ];
?>
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
  <title> الادوات</title>
  <meta name="twitter:title" content="">
  <meta property="og:image" content="assets/img/logo01.jpg">
  <meta name="description" content=" ">
  <meta property="og:type" content="website">
  <meta name="twitter:image" content="assets/img/logo02.png">
  <link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
  <link rel="stylesheet" href="assets/bootstrap/css/bootstrap.min.css">
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Lato:300,400,700">
```

```
        <link rel="stylesheet" href="assets/fonts/ionicons.min.css">
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/css/pikaday.min.css">
        <link rel="stylesheet" href="assets/css/untitled.css">
      </head>

      <body>
        <?php
        include("Nav.php");
        ?>
        <main class="page hire-me-page">
          <section class="portfolio-block hire-me">
            <div class="container" style="width: 1146px;">
              <h2><strong>---- </strong></h2>
              <form action="SearchR3.php" style="width: 100%;max-width: 100%;" method = "POST" enctype = "multipart/form-data">
                <div class="form-group">
                  <div class="form-row">
                    <div class="col-md-6 button offset-xl-0">
                      <p>

                        <label for="Company">Search by <?php echo $_SESSION['UserNameT']; ?> for </label>
                        <input name="iD" type="text" class="form-control" id="iD" value="" >

                      </p>

                    </div>
      <div class="col">
          <div class="col">
```

```
<label for="subject">&amp;nbsp;</label><br>
      </div>

       <input type = "submit" class="form-control" title="search"/>

          </div>
        </div>
       </div>
       <div class="form-row"> </div>

      </form>
     </div>
    </section>
   </main>
   <footer class="page-footer">
    <div class="container">

    </div>
   </footer>
   <script src="assets/js/jquery.min.js"></script>
   <script src="assets/bootstrap/js/bootstrap.min.js"></script>
   <script
src="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/pikaday.min.js"></script>
   <script src="assets/js/theme.js"></script>
  </body>

  </html>
```

SEARCH A

```
<?php
session_start();
include("authorization_1.php");

//set session variables in a $_SESSION global array variable
$_SESSION[ 'UserName' ] = 'UserName';
//$_SESSION[ 'role' ] = 'admin';
//$pid=$_SESSION[ 'pid' ];
?>
<!DOCTYPE html>
<html>

<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
 <title> الادوات</title>
 <meta name="twitter:title" content="">
 <meta property="og:image" content="assets/img/logo01.jpg">
 <meta name="description" content=" ">
 <meta property="og:type" content="website">
 <meta name="twitter:image" content="assets/img/logo02.png">
 <link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
 <link rel="stylesheet" href="assets/bootstrap/css/bootstrap.min.css">
 <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Lato:300,400,700">
 <link rel="stylesheet" href="assets/fonts/ionicons.min.css">
```

```
        <link                                                    rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/css/pikaday.min.css">
        <link rel="stylesheet" href="assets/css/untitled.css">
      </head>

      <body>
        <?php
        include("Nav.php");
         ?>
        <main class="page hire-me-page">
         <section class="portfolio-block hire-me">
          <div class="container" style="width: 1146px;">
            <h2><strong>---- </strong></h2>
           <form action="SearchR23.php" style="width: 100%;max-width: 100%;"
method = "POST" enctype = "multipart/form-data">
            <div class="form-group">
             <div class="form-row">
              <div class="col-md-6 button offset-xl-0">
               <p>

                <label for="Company">Search by <?php echo
$_SESSION['UserNameT']; ?> for </label>
                <input name="iD" type="text" class="form-control" id="iD"
value="" >

               </p>

              </div>
      <div class="col">
          <div class="col">
      <label for="subject">&amp;nbsp;</label><br>
```

```
            </div>

             <input type = "submit" class="form-control" title="search"/>

                </div>
               </div>
              </div>
              <div class="form-row"> </div>

              </form>
             </div>
            </section>
           </main>
           <footer class="page-footer">
            <div class="container">
             <div class="links"><a href="#">About us</a><a href="#">Contact us</a><a href="#">Data</a></div>
             <div class="social-icons"><a href="#"><i class="icon ion-social-facebook"></i></a><a href="#"><i class="icon ion-social-instagram-outline"></i></a><a href="#"><i class="icon ion-social-twitter"></i></a></div>
            </div>
           </footer>
           <script src="assets/js/jquery.min.js"></script>
           <script src="assets/bootstrap/js/bootstrap.min.js"></script>
           <script src="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/pikaday.min.js"></script>
           <script src="assets/js/theme.js"></script>
          </body>

          </html>
```

```
Search History
<?php
session_start();
include("authorization_A.php");

//set session variables in a $_SESSION global array variable
$_SESSION[ 'username' ] = 'username';
$_SESSION[ 'role' ] = 'admin';
$partid=$_SESSION[ 'partid' ];
?>

<?php

include("conn.php");



?>

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
  <title>---------- </title>
  <meta name="twitter:title" content="Data">
  <meta property="og:image" content="assets/img/logo01.jpg">
  <meta name="description" content="NGO Coordination Home Projects contact">
  <meta property="og:type" content="website">
  <meta name="twitter:image" content="assets/img/logo02.png">
```

```
        <link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">

        <link rel="stylesheet" href="assets/bootstrap/css/bootstrap.min.css">
        <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Lato:300,400,700">
        <link rel="stylesheet" href="assets/fonts/ionicons.min.css">
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/css/pikaday.min.css">
        <link rel="stylesheet" href="assets/css/untitled.css">
        <style>
        table {
        font-family: arial, sans-serif;
        border-collapse: collapse;
        width: 100%;
        }

        td, th {
        border: 1px solid #dddddd;
        text-align: left;
        padding: 8px;
        }

        tr:nth-child(even) {
        background-color: #dddddd;
        }
        </style>
        <script>
        function myFunction()
        {
        alert("New record created successfully!"); // this is the message in ""
```

```
}
</script>
</head>
<body>
 <br><br><br><br><br>
  <?php
 include("Nav.php");
   ?>
 <main class="page hire-me-page">
  <section class="portfolio-block hire-me">
    <div class="container" style="width: 1146px;">
      <h2><strong>History </strong></h2>
     <?php
 include("Conn.php");



      #employees
#$sql22 = $pdo->prepare('SELECT * FROM tooltb WHERE Tool = :Tool');
 $sql = "SELECT * FROM historytb ORDER BY id DESC";
    // echo "<br>--++-$-++-)---"+"<br>"+"<bp>"+"<p>---------"+$sql+"<br>";



 $result = $conn->query($sql);

 if ($result->num_rows > 0) {
 echo " <table><tr><th>ID</th><th> Suspicion SQL Injection Attacks</th><th>Date</th> ><th>User</th>
```

```
</tr>";
// output data of each row
  $k=0;
while($row = $result->fetch_assoc()) {

 $k=$k+1;

//Address userF
  $st1=$row["dataF"];

$new = str_replace("-", "", $st1);
 echo "<tr><td>".$row["id"]."</td><td> ".$new ." </td><td>
".$row["DateE"]."</td>
<td>". $row["userF"]." </td></tr>";
}

echo "</table>" ."  ".$temp;



} else {
echo "   <br>".$data[0][1]." <br>";
}
     //$ttt=$data[0][1];
 //$NameP=$decryption_value ($ttt, $ciphering_value, $encryption_key);

 // echo "<br> ttt =  <br>".$sql." <br>33<br> ".de('wrBa')." <br>44<br>";

     ////<?php echo $file_name ."<br>";if ($file_name )echo '<img
src="images/'.$file_name.'" alt="File" width="50">';
 ?>
   </div>
```

```
  </section>
  </main>
 <footer class="page-footer">
  <div class="container">
  <div class="about-me">


    </div>
    </div>
  </div>
 </footer>
 <script src="assets/js/jquery.min.js"></script>
 <script src="assets/bootstrap/js/bootstrap.min.js"></script>
 <script
src="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/pikaday.min.js"></script>
 <script src="assets/js/theme.js"></script>
</body>

</html>
```

SEARCH R3

```
<?php
session_start();
//include("authorization_1.php");
 $UserNameT= $_SESSION['UserNameT'];
```

```
function de($txt)
{
 $ciphering_value = "AES-128-CTR";
 $decryption_key = "1@";

 $decryption_value = openssl_decrypt($txt, $ciphering_value, $decryption_key);
 return $decryption_value;
 $message = "Alert !!";
 echo "<script type='text/javascript'>alert('$message $txt');</script>";
}
//set session variables in a $_SESSION global array variable
$_SESSION[ 'username' ] = 'username';
$_SESSION[ 'role' ] = 'admin';
$partid=$_SESSION[ 'partid' ];
?>

<?php

include("conn.php");



?>

<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
 <title>---------- </title>
```

```
<meta name="twitter:title" content="Data">
<meta property="og:image" content="assets/img/logo01.jpg">
<meta name="description" content="NGO Coordination Home Projects contact">
<meta property="og:type" content="website">
<meta name="twitter:image" content="assets/img/logo02.png">
<link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
<link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
<link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
<link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
<link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
<link rel="stylesheet" href="assets/bootstrap/css/bootstrap.min.css">
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Lato:300,400,700">
<link rel="stylesheet" href="assets/fonts/ionicons.min.css">
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/css/pikaday.min.css">
<link rel="stylesheet" href="assets/css/untitled.css">
<style>
table {
 font-family: arial, sans-serif;
 border-collapse: collapse;
 width: 100%;
}

td, th {
```

```
 border: 1px solid #dddddd;
 text-align: left;
 padding: 8px;
}

tr:nth-child(even) {
 background-color: #dddddd;
}
</style>
  <script>
function myFunction()
{
alert("New record created successfully!"); // this is the message in ""
}
</script>
</head>
<body>
  <br><br><br><br><br>
   <?php
  include("Nav.php");
    ?>
  <main class="page hire-me-page">
    <section class="portfolio-block hire-me">
      <div class="container" style="width: 1146px;">
         <h2><strong>Search Results </strong></h2>
        <?php
  include("Conn.php");
```

```
        // echo '<br><br><br><br>'; echo '<br><br><br><br>'; echo '<br><br><br><br>----------<hr>';
         $iD=$_POST['iD'];

              //like '%%'
        $temp=strpos($iD, "=");
        $temp=intval($temp);
          //$message = "Alert !!".$temp;
          //echo "<script type='text/javascript'>alert('$message $iD');</script>";



              #employees
        #$sql22 = $pdo->prepare('SELECT * FROM tooltb WHERE Tool = :Tool');
         $sql = "SELECT * FROM nametb where iD=$iD";
           //        echo        "<br>--++-$temp-++-)---"+"<br>"+"<bp>"+"<p>---------
"+$sql+"<br>";
        if ($temp>1)
        {
          //$sql = "";
          //$message = "Alert !!  Alert !!  Alert !!";

          include("conn.php");
          //$iD = preg_replace("!\'\"[^a-z0-9]+!i", "-", $iD);
          $new = str_replace("'", "-", $iD);
        $dataF=$new;

        $sql = "INSERT INTO historytb (dataF,userF) VALUES
('$dataF','$UserNameT')";

        if ($conn->query($sql) === TRUE) {
```

```
 echo "New record created successfully. ";
}
else {
 echo "<br><br>Error: " . "<br>" . $conn->error;
}

}
 $result = $conn->query($sql);

 if ($result->num_rows > 0) {
 echo "<table><tr><th>ID</th><th>Name</th><th>Card no</th><th>phon No</th>
 <th>Address </th> </tr>";
 // output data of each row
  $k=0;
 while($row = $result->fetch_assoc()) {

  $k=$k+1;
 $NameP=$row["NameP"];
 $ciphering_value = "AES-128-CTR";
 $decryption_key = "1@";
 $NameP=$decryption_value = openssl_decrypt($NameP, $ciphering_value, $decryption_key);
 //$data=array(array($k,$row["NameP"],$row["password"])) ;
 //$NameP=$decryption_value ($NameP, $ciphering_value, $encryption_key);

 //$NameP=$row["NameP"];
 //$NameP=$decryption_value ("bvb", "AES-128-CTR", "1@");
 $CardNo=$row["CardNo"];
 $CardNo=$decryption_value = openssl_decrypt($CardNo, $ciphering_value, $decryption_key);
```

```
$Phone=$row["Phone"];
$Phone=$decryption_value = openssl_decrypt($Phone, $ciphering_value, $decryption_key);

$Address=$row["Address"];
$Address=$decryption_value = openssl_decrypt($Address, $ciphering_value, $decryption_key);
//$Phone=$decryption_value ($Phone, $ciphering_value, $encryption_key);



//Address

 echo "<tr><td>".$row["iD"]."</td><td> ".$NameP."</td><td> ".$CardNo."</td>
<td>". $Phone." </td><td>". $Address." </td> </tr>";
}

echo "</table>" ;
  echo "<p><a class='btn btn-outline-primary' role='button' href=".$cf."search.php>Back </a>";



} else {
echo " <br> <br><h2> 0 results  </h2> <br>";
  echo "<p>  <br><a class='btn btn-outline-primary' role='button' href=".$cf."search.php> Back </a>";
}
     //$ttt=$data[0][1];
 //$NameP=$decryption_value ($ttt, $ciphering_value, $encryption_key);
```

```
// echo "<br> ttt =  <br>".$sql." <br>33<br> ".de('wrBa')." <br>44<br>";

////<?php echo $file_name ."<br>";if ($file_name )echo '<img src="images/'.$file_name.'" alt="File" width="50">';
?>
</div>
</section>
</main>
<footer class="page-footer">
<div class="container">
<div class="about-me">

</div>

</div>
</footer>
<script src="assets/js/jquery.min.js"></script>
<script src="assets/bootstrap/js/bootstrap.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/pikaday.min.js"></script>
<script src="assets/js/theme.js"></script>
</body>

</html>

Search R23
<?php
session_start();
include("authorization_A.php");
```

```
//set session variables in a $_SESSION global array variable
$_SESSION[ 'username' ] = 'username';
$_SESSION[ 'role' ] = 'admin';
$partid=$_SESSION[ 'partid' ];
?>

<?php

include("conn.php");


?>

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, shrink-to-fit=no">
  <title>---------- </title>
  <meta name="twitter:title" content="Data">
  <meta property="og:image" content="assets/img/logo01.jpg">
  <meta name="description" content="NGO Coordination Home Projects contact">
  <meta property="og:type" content="website">
  <meta name="twitter:image" content="assets/img/logo02.png">
  <link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
  <link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
```

```
        <link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
        <link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
        <link rel="icon" type="image/jpeg" sizes="931x870" href="assets/img/logo01.jpg">
        <link rel="stylesheet" href="assets/bootstrap/css/bootstrap.min.css">
        <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Lato:300,400,700">
        <link rel="stylesheet" href="assets/fonts/ionicons.min.css">
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/css/pikaday.min.css">
        <link rel="stylesheet" href="assets/css/untitled.css">
        <style>
        table {
         font-family: arial, sans-serif;
         border-collapse: collapse;
         width: 100%;
        }

        td, th {
         border: 1px solid #dddddd;
         text-align: left;
         padding: 8px;
        }

        tr:nth-child(even) {
         background-color: #dddddd;
        }
        </style>
         <script>
```

```
function myFunction()
{
alert("New record created successfully!"); // this is the message in ""
}
</script>
</head>
<body>
 <br><br><br><br><br>
  <?php
 include("Nav.php");
  ?>
 <main class="page hire-me-page">
  <section class="portfolio-block hire-me">
   <div class="container" style="width: 1146px;">
     <h2><strong>Data </strong></h2>
     <?php
 include("Conn.php");



 // echo '<br><br><br><br>'; echo '<br><br><br><br>'; echo
'<br><br><br><br>------------------------<hr>';
 $iD=$_POST['iD'];

      //like '%%'
$temp=strpos($iD, "=");
//$message = "Alert !!".$temp;
  //echo "<script type='text/javascript'>alert('$message $iD');</script>";
if ($temp>1)
{
```

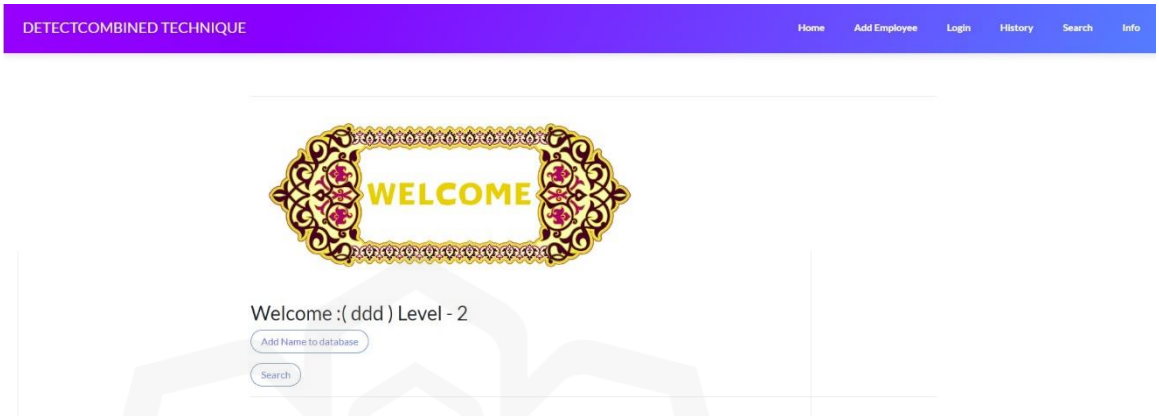
```
 $message = "Alert !!";
 echo "<script type='text/javascript'>alert('$message $iD');</script>";

}

     #employees
#$sql22 = $pdo->prepare('SELECT * FROM tooltb WHERE Tool = :Tool');
 $sql = "SELECT * FROM nametb where iD=$iD";
    echo "<br>-------"+"<br>"+"<bp>"+"<p>---------"+$sql+"<br>";
 $result = $conn->query($sql);

 if ($result->num_rows > 0) {
 echo "<table><tr><th>ID</th><th>Name</th><th>Phone</th>
<th>Address</th>
 </tr>";
 // output data of each row
 while($row = $result->fetch_assoc()) {

  $NameP = openssl_decrypt($row["NameP"], $ciphering_value,
$decryption_key);

 echo
"<tr><td>".$row["iD"]."</td><td>".$row["NameP"].$NameP."</td><td>
".$row["CardNo"]."</td>
  <td>". $row["Phone"]." >>".$temp."</td><td> </tr>";
 }

 echo "</table>" ."results ".$sql." --".$temp;

 } else {
 echo "0 results ".$sql;
```

```
}

        ////<?php echo $file_name ."<br>";if ($file_name )echo '<img src="images/'.$file_name.'" alt="File" width="50">';
        ?>
          </div>
        </section>
        </main>
       <footer class="page-footer">
        <div class="container">
        <div class="about-me">


          </div>
          <div class="links"><a href="about us.php">About us</a><a href="#">Contact us</a></div>
          <div class="social-icons"><a href="#"><i class="icon ion-social-facebook"></i></a><a href="#"><i class="icon ion-social-instagram-outline"></i></a><a href="#"><i class="icon ion-social-twitter"></i></a></div>
        </div>
       </footer>
       <script src="assets/js/jquery.min.js"></script>
       <script src="assets/bootstrap/js/bootstrap.min.js"></script>
       <script src="https://cdnjs.cloudflare.com/ajax/libs/pikaday/1.6.1/pikaday.min.js"></script>
        <script src="assets/js/theme.js"></script>
      </body>

      </html>
```

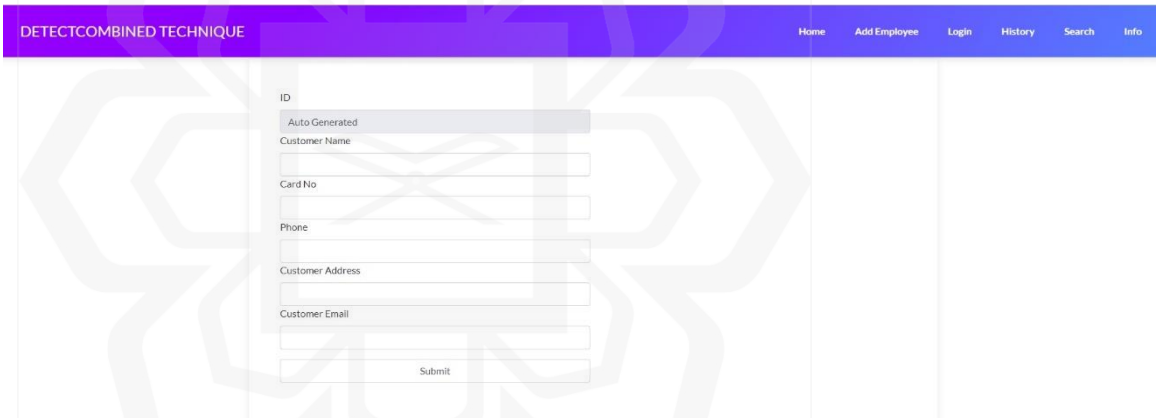
APPENDIX II

1- Add Name to Database / User level 2

iv.



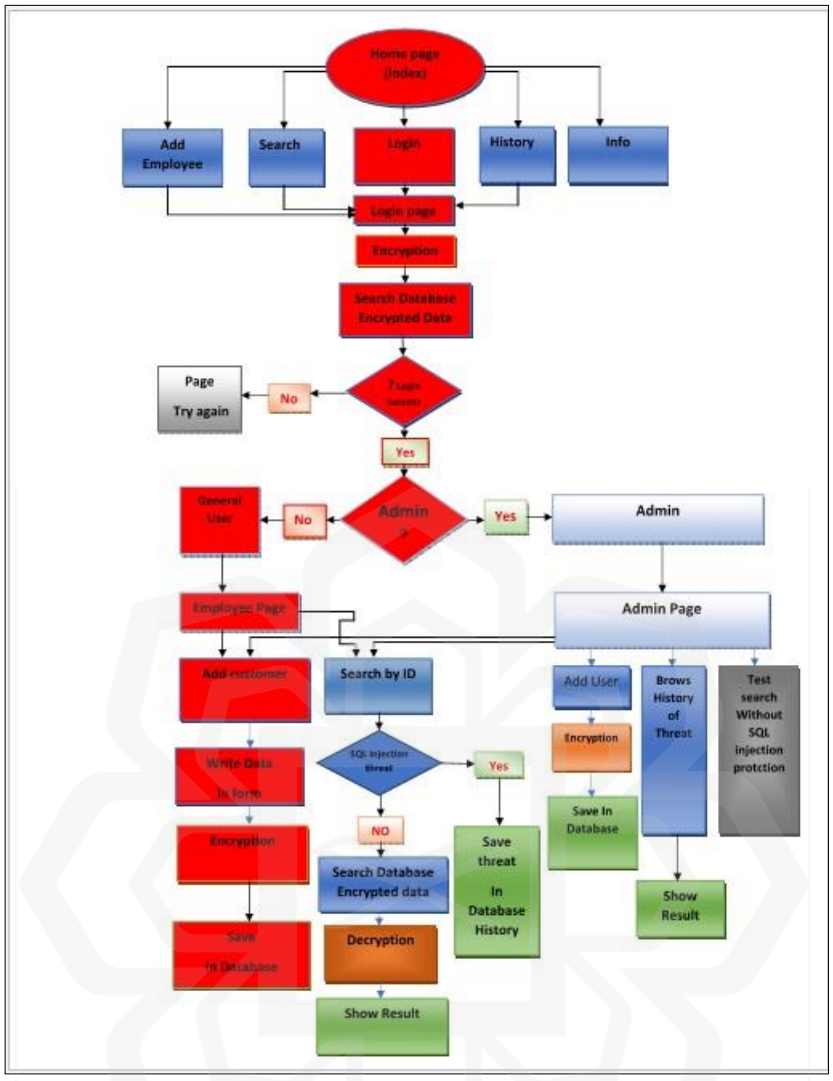
v.



vi.

vii.

viii.

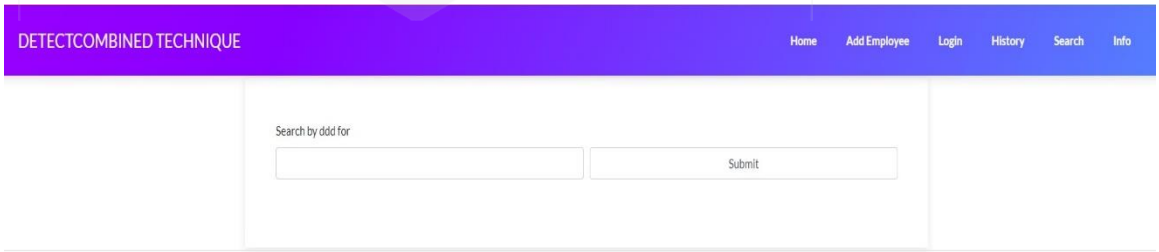


ix.

x.

2- Search / User level 2

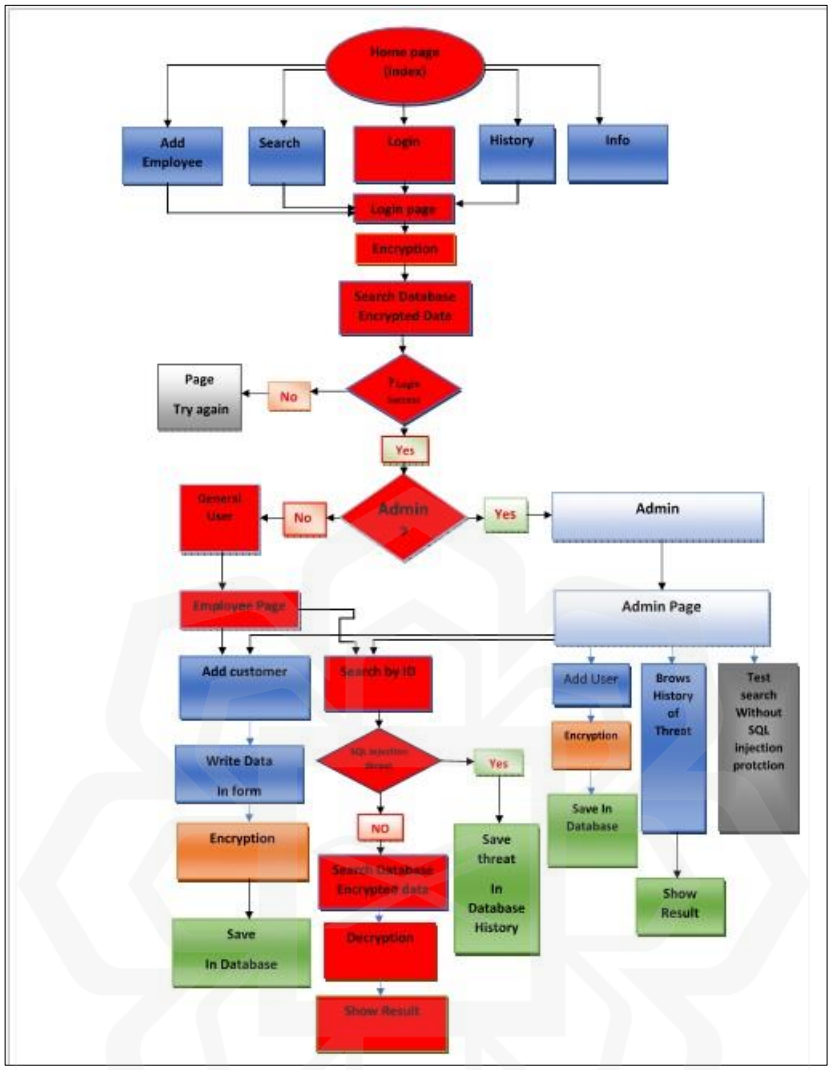
xi.



xii.

xiii.

xiv.


Home page (index)
Add Employee
Search
Login
History
Info
Login page
Encryption
Search Database Encrypted Data
Page Try again
No
Yes
General User
No
Admin ?
Yes
Admin
Employee Page
Admin Page
Add customer
Search by ID
Add User
Brows History of Threat
Test search Without SQL injection protction
Encryption
Write Data In form
Yes
Save In Database
Encryption
NO
Save threat In Database History
Show Result
Save In Database
Decryption
Show Result