International Journal of Engineering Trends and Technology
ISSN: 2231–5381 / https://doi.org/10.14445/22315381/IJETT-V71I8P219

Volume 71 Issue 8, 216-224, August 2023

Original Article

An Approach to Detect and Prevent SQL Injection and XSS Vulnerability in the Web Application

Shekhar Disawal[1], Ugrasen Suman[2]

[1,2]*School of Computer Science & IT, Devi Ahilya University, Indore, M.P., India*

[1]*Corresponding Author : shekhar.disawal@gmail.com*

Received: 10 April 2023 Revised: 16 June 2023 Accepted: 13 July 2023 Published: 15 August 2023

Abstract *- SQL Injection (SQLI) and Cross-Site Scripting (XSS) are commonly exploited vulnerabilities in web applications, particularly those connected to sensitive data like banking, finance, and e-commerce. These attacks allow the attackers to gain unauthorized access to the system and manipulate or delete crucial data. The attack is carried out by injecting malicious SQL statements into a query through an unvalidated input field. As a result, it is crucial to find effective solutions to detect and prevent these vulnerabilities in web applications. Although several methods have been proposed by researchers, existing solutions have limitations and inefficiencies in protecting against web attacks. In this paper, we propose a Web Vulnerability-Detection Prevention Methodology (WV-DPM) that can effectively detect and prevent SQL injection and XSS attacks. To evaluate the effectiveness of our proposed methodology, we have implemented it and compared it with existing methodologies.*

Keywords *- SQL injection, Prevention, Detection, Vulnerability, Web application.*

1. Introduction

Data is a very important part of any business. Today, most organizations want to increase profit and improve their relationships and customer communication using web applications [1]. The database is the backbone of web applications such as Oracle, MSSQL, MySQL, and MS Access [2]. Most users use their online platforms for business and other purposes. As internet usage grows, there is a need to provide secure and crucial data communications. In January 2018, the population of internet users saw a significant boost, with a reported increase to 3.48 billion individuals. This figure continued to climb in January 2019, reaching a staggering 4.39 billion users, representing a notable surge from the previous year [3].

With the increased popularity of web applications in various fields, such as social media, finance, and health, web vulnerabilities have become a serious concern. Many unwanted activities occur among users as a result of insecure web applications due to flaws or loopholes in web applications. Web security flaws can threaten personal details and other precious things. A user tries to request a web server using HTML forms, URLs, or other fields where data can be entered. The unfiltered form permits SQL injection via users. The database continuously processes the form data without verification [4].

Mostly, the attackers have knowledge of web application development, and injecting malicious script into a web page viewed by a remote location on the website is known as a Cross-Site scripting attack. The most common online application security issue is cross-site scripting (XSS), making web application security an essential concern [5]. SQL injection and XSS attacks are classified into several categories, which are explained in subsequence subsections.

1.1. SQL Injection Attack

An SQL injection attack is launched when a hacker exploits a vulnerability in a website's login form to gain access to or alter the database. SQL injection vulnerabilities allow attackers to directly submit commands to the online application's database, compromising privacy and effectiveness [6]. SQL Injection can be six types and how they are executed, but also in the data they gather, the data they modify, the commands they run, and the services they disrupt [7]. SQL injection attacks can take many forms, but some of the most common ones include Boolean, Error, Union, Like, Batch Query, and Encoded.

A Boolean-based SQL injection attack is a type of cyber-attack where an attacker injects an SQL query into a vulnerable application to obtain information from a database. The attacker can manipulate the query to force the application to respond with a different output based on whether the query evaluates to true or false. The attacker can then deduce whether the payload used in the attack returns a true or false value, even though no actual data from the database is returned. This attack is often time-consuming since the attacker needs to go through the database one character at a time. An error-based SQL injection attack works by putting invalid data into the query, which causes a database error. The attacker can then look for errors made by the database and use those errors

to learn more about how to use the SQL query to change the database. A Union-based SQL injection attack takes advantage of the UNION operator in SQL, which is used to join two SQL statements or queries. The attacker injects another query in place of plain text and uses the UNION keyword at the beginning of the query. This causes the database to return both desired and intended results. A like-based SQLI attack is a sort of cyber-attack in which attackers utilize flaws in an application's search capability to impersonate a specific user. The attacker can inject input, which includes the LIKE operator, to change the behaviour of the query. A Batch SQL injection query attack injects a malicious query into the database server by using multiple queries at the same time. The attacker can terminate the original query with a semicolon and inject a malicious query into the database server. Encoded-based SQL Injection attack uses encoded input to evade detection by security measures such as firewalls or intrusion detection systems.

1.2. Cross-Site Scripting (XSS) Attack

XSS is a type of cyberattack where hackers insert harmful code into web pages displayed by a reputable web application. This deceitful content appears trustworthy and is treated as normal content by the web application, making it difficult to detect. Users might be sent to harmful websites without their permission if a website has been compromised and subsequent malware has been installed. The attacker may also hijack the entire user session, steal login information, and access sensitive data. There are three types of XSS vulnerabilities: Reflected, Stored, and Document Object Model (DOM)-based. When dynamic material on a website (often JavaScript) is manipulated by an attacker and then executed, this is called a DOM-based vulnerability. Web applications are susceptible to stored XSS flaws if they save potentially harmful user data for later processing. Given that an attacker may use this flaw to alter any data in the database, it is one of the most severe forms of XSS vulnerability [1]. Reflected XSS vulnerabilities are a type of security flaw that differs from other types of XSS vulnerabilities because they target users who view or load a harmful web address.

Many different tactics have been recorded in the research that tries to lessen the escalating threats posed by these attacks; nevertheless, very few of these strategies have been able to address the entire scope of the problem. Numerous security solutions have been put forward to stop unauthorized access to data and information, but attackers keep making new security holes that can be used [8,27]. It is crucial to develop new approaches to detect and prevent cyber threats as they become more advanced and sophisticated.

The structure of the paper is as follows Section 1 describes introductory aspects of SQLI and XSS attacks. Section 2 presents the related work for web application vulnerability detection and prevention. Section 3 explains the proposed Web Vulnerability-Detection Prevention Methodology (WV-DPM). Section 4 represents the experiment work. The discussion is in Section 5, and the conclusion is in Section 6.

2. Related Work

SQL injection (SQLI) and cross-site scripting (XSS) are two types of assaults that have led to the development of a number of different methods for their detection and prevention. By utilizing data encryption methods, PHP escape methods, pattern-matching strategies, and instruction set randomization, SQL injection vulnerabilities and XSS attacks have been largely mitigated. Static, dynamic, and hybrid detection are the core components of the majority of SQL injection detection systems. Static detection, or "white box testing," is the practise of using static analysis to find bugs and ensure that code is proper. We added support for spotting tautology attacks in the white-box test. Although useful, this method is limited in that it can only detect certain assaults [8].

Dynamic detection refers to the process of either performing dynamic penetration testing or generating analytical models on-the-fly for web-based applications. This technique involves parsing SQL statements into syntax trees and applying stain analysis to identify any potential SQL injection attacks [27]. In order to discriminate between genuine SQL queries and malicious ones, a better pattern-matching approach was recommended for a signature-based SQLI attack identification framework. This would allow for the differentiation between legitimate and malicious SQL queries. [10]. A mapping model was proposed for SQLI identification and avoidance [11].

Hybrid detection employs pattern matching to identify and obstruct SQL injection attempts, with AMNESIA being a prime example. This technique involves statically analyzing the web application to construct a model of the SQL queries. Afterwards, it observed dynamic queries. Non-compliant queries are prohibited by SQLIA [12]. Another method for detecting and combating SQL injection is based on URL-SQL mapping. Several unidentified factors impact the execution of a SQL statement; therefore, they extracted the specified URL and SQL query to construct a request-to-query mapping model. If invariants in the regular state of the web application are not completely extracted, it will give false alarms and false negative results [13].

The SHA-1 hashing method is a security measure that can prevent SQL injection attacks from affecting batch queries. This method involves retrieving attribute values for queries from stored inputs and using the SHA-1 hashing algorithm to encode them. Before carrying out the task, any new input is converted into a hash and compared with the hash of the previously stored input. The query is denied if the hashed inputs are identical [14]. The Boyer-Moore algorithm was introduced as a means of identifying and preventing SQL injection attacks in input values. A hybrid approach uses the best parts of static and dynamic approaches to find possible vulnerabilities. The static

approach checks the SQL query during the writing stage to identify any flaws, while the dynamic approach checks queries during runtime. The algorithm matches the input query with stored queries and compares the results with the expected valid query. If a vulnerability is detected in a secure system, the query is rejected. On the other hand, the dynamic approach examines incoming queries using an algorithm that detects vulnerabilities and discards them. The proposed approach aims to enhance the security of databases by preventing SQL injection attacks [15].

XSS is the most common form of web application vulnerability. Wherever a web application takes user input, such a vulnerability can potentially insert malicious code. Countless strategies and methodologies have been implemented to sniff out XSS vulnerabilities lurking in source code. One such method is the HTML context-sensitive technique, which employs a combination of taint analysis and defensive programming to pinpoint XSS vulnerabilities that may exist in the source code of PHP-based applications [28].

Despite the continuous evolution of web applications, web vulnerabilities remain an ongoing challenge, demanding ever-vigilant attention from security experts. We identified some limitations in existing SQLI and XSS vulnerability detection and prevention approaches. Limited coverage may not be able to detect or prevent all possible attack scenarios. The majority of mitigation solutions may merely clean input data or escape characters, which may not be sufficient to prevent attacks. Certain detection and prevention technologies may cause considerable performance overhead in the application, affecting the user experience and making them unsuitable for high-traffic applications.

3. Web Vulnerability – Detection Prevention Methodology

We have presented a Web Vulnerability-Detection Prevention Methodology (WV-DPM) for detecting and preventing various types of SQLI and XSS attacks. The patterns of each assault are analyzed, and potential countermeasures are developed based on these analyzed patterns. Fig.1 presents the system architecture of the WV-DPM. The model has six different phases, i.e. planning & scope, asset discovery, attack detection, attack simulation & exploitation, remediation of risk, and analysis & reporting.

The WV-DPM model is a step-by-step approach for identifying web vulnerabilities, with a particular focus on SQLI and XSS attacks. This model can be highly beneficial for developers who want to create secure web applications. We have designed an algorithm that utilizes the Rabin-Karp string-matching algorithm and developed a filter function implemented when a query is submitted. This novel feature ensures that malicious inputs are not stored in the database. The filter function blocks any malicious content and notifies the user of the illegal activity. The overall effect of this method is to make web applications safer and more resistant to cyberattacks.

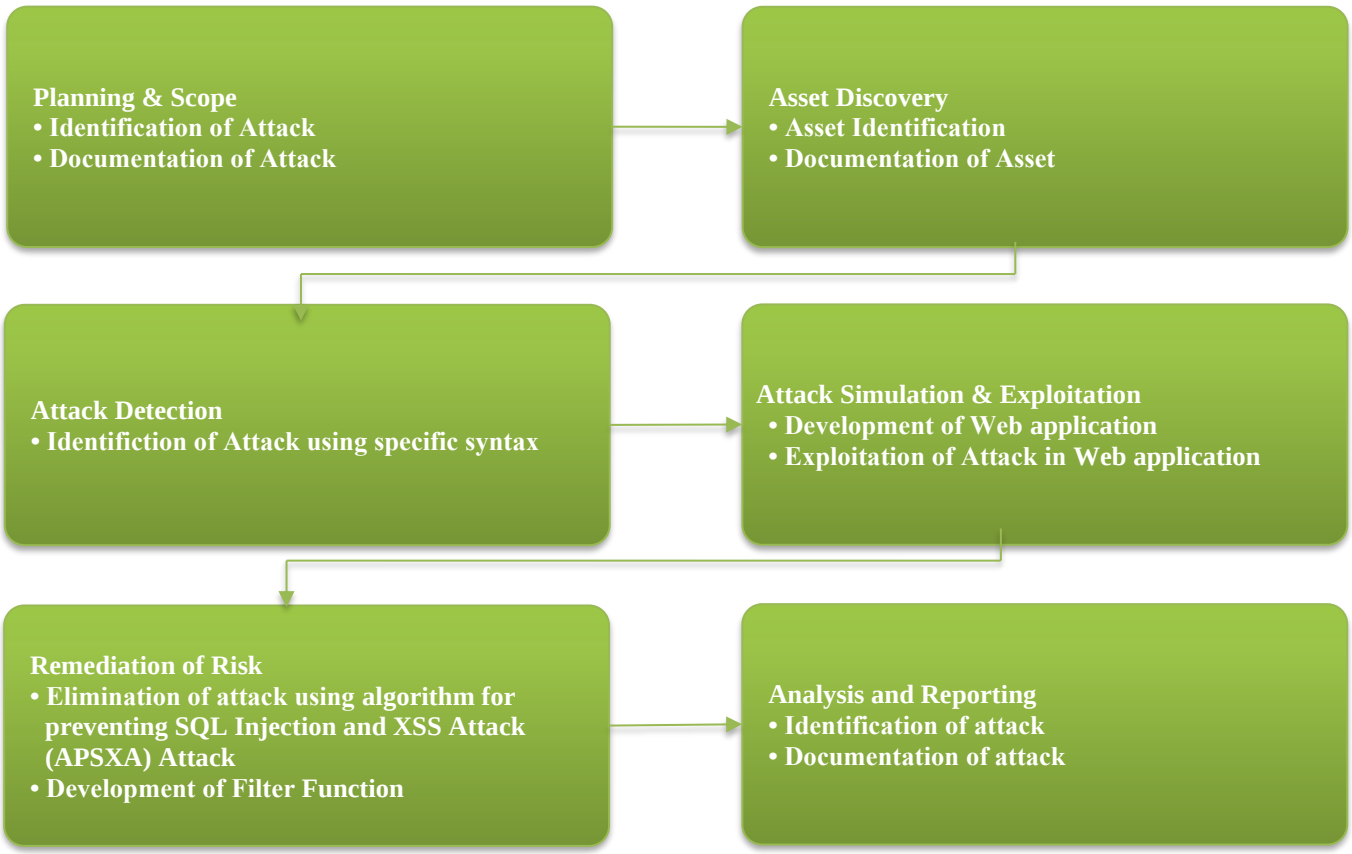

Fig. 1 WV-DPM

A detailed description of each phase is as follows:

3.1. Planning & Scope

Planning and scope refers to the process of identifying, assessing, and addressing potential security risks and vulnerabilities in web applications. The scope of web vulnerability planning typically involves identifying the assets and resources that need to be protected, such as sensitive data or functionality that could be exploited by attackers. We have classified web vulnerabilities and identified SQLI and XSS attacks [17].

SQL injection patterns composed utilizing special characters and keywords are presented in Table 1 and Table 2 [18]. The malicious programs used to launch various attacks are built from these characters and keywords, which are then used in their final form. If these injection codes can be detected and recognized according to the character and keywords mentioned in Table 1 and Table 2, then identifying and preventing these types of attacks would be considerably easier. Table 3 comprises the injection codes that are used by all forms of attacks [18].

Table 1. SQL-Injection code composed using special characters

S.No.	Character	Detail
1	’	For indicating a string of characters
2	%	An Indicator for Wildcard Attributes
3	;	Query ending operator
4	+ or ||	Concatenate Strings
5	=	Assignment Operator
6	-- or #	Single line comment
7	/* ….. */	Multiple line comment
8	>,<, >=,<=, ==, != or <>	operators for comparison

3.2. Asset Discovery

It involves identifying and documenting all the assets, such as web applications, databases, and servers within an organization's infrastructure accessible from the internet.

Table 2. SQL-Injection code composed using keywords

S.No.	Keyword	Detail
1	UNION	Utilized for Union-based injection attack.
2	OR	Utilized for Boolean-based injection attack.
3	DROP	Utilized for the purpose of deleting the whole database table.
4	DELETE	Utilized for removing information from a database table.
5	TRUNCATE	Utilized for clearing out information from a specified table in a database.
6	SELECT	Utilized for retrieving records from a database table.
7	UPDATE	Utilized for modification of records in a database table.
8	INSERT	Utilized for adding records in a database.
9	Like	Utilized in conjunction with the% to choose a record that contains a certain string pattern.
10	CONVERT	Utilized error-based SQL injection to induce the database server to show some warning messages.

Table 3. Various injection code syntaxes with the prevalent attack patterns

S.No.	Type of Injection Assault	Recurring Style	Example
1	Boolean SQLI	’ OR ‘…’=|>|>=|<|<=|<>|!= ‘…’ ;#	SELECT * FROM test WHERE name = '' OR 1=1--';
2	Union SQLI	’ union select * from ….. ;#	‘UNION.SELECT ALL 1, database(),3,4-- -
3	Error-based SQLI	’ … convert (|averag(|round(…)	‘ OR 1=1 GROUP BY CONCATSD_WS (‘-’, version(), FLOOR (rand (0)*2)) having max(0)#
4	Batch Query SQLI	’; drop| delete|insert|update|select * ……;#	'; SELECT * FROM test; --
5	Like-based SQLI	’ OR…. Like ’ ….%’; #	' OR 1=1 AND user_name LIKE '%h%'
6	Encoded SQLI	& # x79 & # x78 & # x32 & # x82	%27%20OR%201=1%20--%20
7	Cross-Site Scripting	<script> ….. ’ …….; < /script>	< img src ="javascript: alert (' you are under web attack')">

Asset discovery helps security professionals to identify all the potential attack surfaces that malicious actors could target. By knowing the assets available in the infrastructure, we can prioritize their efforts and focus on securing the most critical assets.

3.3. Attack Detection

In this phase, we have identified types of patterns to detect SQLI and XSS attacks. The process description is as follows:

The methods listed for identifying SQL injection and XSS attacks include examining specific patterns in user input, such as the presence of certain keywords or characters. Different types of attacks can be identified by the specific syntax used, such as Boolean, Union, Error, Batch query, Like-based SQL, Encoded SQL Injection, and XSS attacks. The following methods can be used to identify the various forms of SQL injection and XSS attacks:

3.3.1. Boolean-Based SQLI Attack

Table 3 illustrates that most Boolean-based SQL injection strings include a single quotation ('), OR, and a true assertion, such as "y" and "5+3" = "10".

3.3.2. Union-Based SQLI Attack

The majority of union-based SQLI strings begin with a single quotation ('), then a UNION keyword, followed by the SQL keyword SELECT, one or more identifiers, the SQL keyword FROM, one or more identifiers, and finally a semicolon (;) with a hash (#). To illustrate, consider the example: "union select * from users; #.

3.3.3. Error-Based SQLI Attack

User input that begins with a single quotation character (') and continues with zero or more SQL functions is likely to be of this type. We can see this in the expressions 121' convert (float, 'xyz'), ‘ A' avg ('&!%$#@*'), and 'round ('xyz', 2).

3.3.4. Batch Query SQLI Attack

Valid SQL statements begin with a single quote (') and terminate with a semicolon (;) and a hash (#). As an illustration, consider the example: abc'; delete * from Icstable; # or '; drop table Ics; #.

3.3.5. Like-Based SQLI Attack

According to Table 3, like-based SQL injection attacks can be spotted when the input string contains a single quotation (') followed by the logical operator OR, one or more identifiers, the SQL term LIKE, a single quotation, the wildcard operator (%), a single quotation, and a semicolon with a hash. Example: "OR uname LIKE "D%"# and "OR pwd LIKE "%4%';#".

3.3.6. Encoded-Based SQL Injection

It involves encoding malicious SQL code into non-readable characters, such as ASCII or Unicode. This can bypass input validation checks, allowing attackers to execute SQL code. The injected code is encoded using ASCII encoding and translates to 'admin'. Example: SELECT * FROM Ics WHERE uname = 'admin';.

3.3.7. Cross-Site Scripting (XSS) Attack

As in <script>alert('XSS');</script>, the input string has a JavaScript open tag "<script>" followed by zero or more characters and/or a single quotation ('). If an XSS attack were to be encoded, a JavaScript opening tag "script>" would be succeeded by one or more ASCII codes, hexadecimal numbers, HTML names, or HTML numbers that represent characters, including the possibility of a single quote ('), as exemplified by the input string <script> alert(&#24; XSS &#24;); < /script>.

3.4. Attack Simulation and Exploitation

Attack simulation and exploitation are important aspects of testing the security of web applications. In this context, attack simulation refers to the process of simulating attacks that a malicious hacker might use to exploit vulnerabilities. Exploitation refers to the actual execution of an attack to take advantage of the vulnerability. We have created a vulnerable web application to simulate a SQL injection and XSS attack. We did manual testing for each type of SQLI and XSS attack mentioned in Table 3 using a different set of characters and phrases same as the attacker’s mind.

3.5. Remediation of Risk

As vulnerabilities have been identified and prioritized, remediation is carried out to eliminate or mitigate them. This may involve applying patches, updating software, reconfiguring security controls, or redesigning the web application architecture.

At this phase, we have developed an algorithm for preventing SQLI and XSS Attacks (APSXA) and developed a filter function to block the malicious input. The algorithm is described as follows.

Algorithm: Algorithm for preventing SQLI and XSS Attacks (APSXA).

Input: I: an array of user input values, each from a form text field.

Output: Blocks the unauthorized user's access, resets HTTP request and gives a warning message or grants access.

Description: The function takes in user input values from each form text field and sums them up to create a single string I. The function then converts I from ASCII to a string format and checks it for various security vulnerabilities using the following sub-functions:

A1 = check_boolean_based_SQLI: checks for boolean-based SQLI
A2 = check_union_based_SQLI: checks for union-based SQLI

A3 = check_error_based_SQLI: checks for error-based SQLI
A4 = check_batch_query_SQLI: checks for batch query SQLI
A5 = check_like_based_SQLI: checks for like-based SQLI
A6 = check_Encoded_based_SQLI: checks for encoded-based SQLI
A7 = check_Xss: checks for cross-site scripting (XSS)

If any sub-functions return true, the function blocks the user's access, resets the HTTP connection, and displays an alert message. Else, the function grants the user access.

Protecting a website from SQL injection and XSS assaults is the primary purpose of the filter() method. Data was sent to the function through the POST method via a web form. In order to thwart encoded injection attempts, the function first changes any ASCII strings it finds in the incoming data.

If the input data is not empty, the function then calls other functions to check for specific forms of attack, such as Boolean-based SQLI, Union-based SQLI, Error-based SQLI, Batch query SQLI, Like-based SQLI, Encoded-based SQLI, and XSS. The outcomes of these checks are represented as variables A1, A2, A3, A4, A5, A6 and A7, respectively.

Filter Function: Check variable types A1, A2, A3, A4, A5, A6 and A7.
Input: A string input to check for A1/A2/A3/A4/A5/A6/A7.
Output: Indicating whether the input contains A1, A2, A3, A4, A5, A6 and A7 or not.
Description: Define injection patterns, input operators, and relational operators.

For each injection pattern (A1/A2/A3/A4/A5/A6/A7) in the injection patterns array:

a. Check if the input contains the injection pattern using the Rabin-Karp Search function.
b. If the injection pattern is found:
i. If this is the first injection pattern, check if the input contains any input operators.
1. If no input operators are found, set the result to false and break the loop.
ii. If this is the third injection pattern, check if the input contains any relational operators.
1. If no relational operators are found, set the result to false and break the loop.
iii. If this is the last injection pattern, set the result to true.
iv. Update the input by removing the injection pattern using the slice and end functions.
c. If the injection pattern is not found, set the result to false and break the loop.
Return the result.

If any of these checks return positive results, indicating an attack has been found, the function will stop the user, resume the HTTP request, and show an alert message. If not, the function will provide access to the user.

3.6. Analysis and Reporting

Web application vulnerability analysis and reporting involve identifying, categorizing, and prioritizing potential security weaknesses in a web application. We have performed manual testing for SQLI and XSS attacks and identified that they require a unique set of characters and phrases that hackers must exploit to carry out their malicious activities. The input field in the web application should be properly validated. We have used a string-matching algorithm in the submit button and stored malicious content in a database table record. It is shown in Fig. 2: The database attributes such as, including types of attacks, input strings field, time stamps record, and message status, were all captured. Through this process, we can analyze attacks and take safe steps to secure our web application.

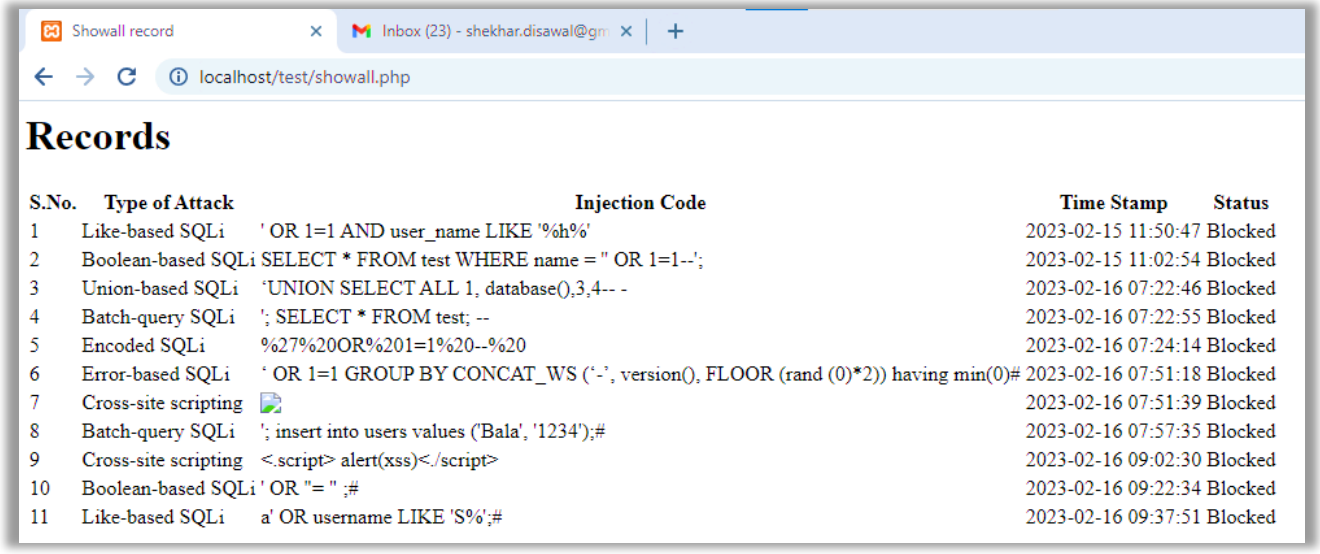

Fig. 2 Attack detection interface showing the records

Table 4. Comparative analysis of existing techniques and the proposed approach

Methodology	Reference	Types of attack						
		Boolean SQLI	Like SQLI	Union SQLI	Error SQLI	Batch query SQLI	Encoded SQLI	Cross-site scripting
Data encryption algorithm	K. D'silva et. al. [19]	Y	Y	Y	Y	Y	Y	N
	U. Upadhyay et al. [20]	Y	Y	Y	Y	Y	Y	N
	Q. Temeiza et. al. [14]	Y	Y	Y	Y	Y	Y	N
	A. John et al. [21]	Y	Y	Y	Y	Y	Y	N
Instruction set randomization	C. Ping et al. [22]	Y	Y	Y	Y	Y	N	N
	G. Buja et. al. [23]	Y	Y	Y	Y	Y	N	N
String matching algorithm	A. Ghafarian [24]	Y	Y	Y	Y	Y	N	Y
	A.Ramesh et al. [29]	Y	Y	Y	Y	Y	N	Y
	M.A. Prabakar et. al. [26]	Y	Y	Y	Y	Y	N	Y
WV-DPM Model		Y	Y	Y	Y	Y	Y	Y

4. Experiment

The experiment of various attacks performed on the Apache Web Server. The Apache XAMPP Server web server and the PHP programming language were used to implement the WV-DPM paradigm. Due to its broad use and acceptance in developing web-based applications that depend on database integration, PHP was particularly selected as the server-side programming language.

The assault was launched using Windows operating systems on localhost. The browsers used to initiate the attack were Opera (version 74.0.3911.107), Google Chrome (version 109.0.5414.122), Mozilla Firefox (version 72.0), and Microsoft Edge (version 110.0.1587.46). The target database was saved on the MySQL database. The test was carried out on the localhost XAMPP server.

Using the test strategy laid out in Table 3, an effort was made to submit several documented instances of SQL injection and cross-site scripting attacks. Different types of attack input strings are included in the test cases that make up the test plan. The web application's input fields were used to supply the input strings. The request would be blocked when an attack is detected. It was recorded in a database table illustrated in Fig. 2 that the types of attacks, input strings code submitted, time stamps, and attack status were all captured. Based on the results of the different attempts, the proposed approach found and stopped all of the attacks.

5. Discussion

The proposed technique was compared to existing methodologies in the field of web application security. Specifically, four articles were analyzed that focused on data encryption techniques, two articles on instruction set randomization, and three articles on string matching algorithms.

The results of this comparative analysis are summarized in Table 4. It was found that all six types of SQL injection attacks can be prevented by existing techniques that use data encryption algorithms, although XSS attacks may still be possible. On the other hand, the suggested WV-DPM approach was proven effective against all six distinct forms of SQL injection attacks, including XSS and encoded injection attempts. It also noted that all other known solutions have their limitations and drawbacks, which makes the proposed WV-DPM methodology a highly promising approach for improving the security of web applications. These findings suggest that WV-DPM can be an effective technique for preventing web vulnerabilities and can help developers create more secure web applications.

6. Conclusion

The WV-DPM model proposed in this paper offers an effective solution to detect and prevent SQL injection and XSS attacks in web applications. By studying the different types and patterns of these attacks in a systematic way, the model would be able to find and stop them, log the attack in the database, and send out a "blocked" message. The comparison with existing techniques shows that WV-DPM is a more efficient approach in stopping all six types of SQL injection attacks, including cross-site scripting and encoded injection attacks.

The results of this research demonstrate the importance of continuously improving web application security to prevent cyber-attacks. With the increasing number of web-based applications and services, it is crucial to have effective and efficient methods in place to identify and prevent potential vulnerabilities. The WV-DPM model significantly contributes to this area of research by providing a comprehensive solution that can help developers create more secure web applications and reduce the risk of vulnerabilities being exploited.

References

[1] Sudhakar Choudhary, Arvind Kumar Jain, and Anil Kumar, "A Detail Survey on Various Aspects of SQLIA," *International Journal of Computer Applications*, vol. 161, no. 12, pp. 34–39, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[2] Mazoon Al Rubaiei et al., "SQLIA Detection and Prevention Techniques," *9th International Conference on System Modeling & Advancement in Research Trends*, pp. 115–121, 2020. [CrossRef] [Publisher Link]

[3] Acunetix Web Application Vulnerability Report 2019, 2019. [Online]. Available: https://www.acunetix.com/white-papers/acunetix-web-application-vulnerability-report-2019/

[4] Qi Li et al., "LSTM-based SQL Injection Detection Method for an Intelligent Transportation System," *IEEE Transactions on Vehicular Technology*, vol. 68 no. 5, pp. 4182-4191, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[5] Shashank Gupta, and B. B. Gupta, "XSS-Secure as a Service for the Platforms of Online Social Network-Based Multimedia Web Applications in Cloud," *Multimedia Tools and Applications*, vol. 77, no. 4, pp. 4829-4861, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[6] Peng Tang et al., "Detection of SQL Injection Based on Artificial Neural Networks," *Knowledge-Based Systems*, vol. 190, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[7] Da-Yu Kao, Chung-Jui Lai, and Ching-Wei Su, "A Framework for SQL Injection Investigations: Detection, Investigation, and Forensics," *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2838–2843, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[8] Gary Wassermann et al., "Static Checking of Dynamically Generated Queries in Database Applications," *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 4, pp. 14-es, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[9] Aqsa Afroz et al., "An Algorithm for Prevention and Detection of Cross-Site Scripting Attacks," *SSRG International Journal of Computer Science and Engineering*, vol. 7, no. 7, pp. 8-18, 2020. [CrossRef] [Publisher Link]

[10] Benjamin Appiah, Eugene Opoku-Mensah, and Zhiguang Qin, "SQL Injection Attack Detection Using Fingerprints and Pattern Matching Technique," *8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 583–587, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[11] Rathod Mahesh Pandurang, and Deepak C. Karia, "A Mapping-Based Model for Preventing Cross-Site Scripting and SQL Injection Attacks on Web Application and its Impact Analysis," *1st International Conference on Next Generation Computing Technologies (NGCT)*, pp. 414–418, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[12] William G J Halfond, and Alessandro Orso, "AMNESIA: Analysis and Monitoring For Neutralizing SQL-Injection Attacks," *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pp. 174–183, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[13] Zeli Xiao et al., "An Approach for SQL Injection Detection Based on Behavior and Response Analysis," *IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, pp. 1437–1442, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[14] Qais Temeiza, Mohammad Temeiza, and Jamil Itmazi, "A Novel Method for Preventing SQL Injection using SHA-1 Algorithm and Syntax-Awareness," *Joint International Conference on Information and Communication Technologies for Education and Training and International Conference on Computing in Arabic (ICCA-TICET)*, pp. 1-4, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[15] Geogiana Buja et al., "Detection Model for SQL Injection Attack: An Approach for Preventing a Web Application from the SQL Injection Attack," *IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, pp. 60–64, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[16] Dian kurnia, Hendry, and Muhammad Syahputra Novelan, "The Forensic Approach Uses Snort from SQL Injection Attacks on the Server," *International Journal of Computer Trends and Technology*, vol. 68, no. 6, pp. 51-56, 2020. [CrossRef] [Publisher Link]

[17] Shekhar Disawal, and Ugrasen Suman, "An Analysis and Classification of Vulnerabilities in Web-Based Application Development," *8th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 782-785, 2021. [Google Scholar] [Publisher Link]

[18] Oluwakemi Christiana Abikoye et al., "A Novel Technique to Prevent SQL Injection and Cross-Site Scripting Attacks Using Knuth-Morris-Pratt String Match Algorithm," *EURASIP Journal on Information Security*, vol. 2020, no. 14, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[19] Karis D'silva et al., "An Effective Method for Preventing SQL Injection Attack and Session Hijacking," *IEEE International Conference on Recent Trends in Electronics Information & Communication Technology (RTEICT)*, pp. 697–701, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[20] Utpal Upadhyay, and Girish Khilari, "SQL Injection Avoidance for Protected Database with ASCII using SNORT and Honeypot," *International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, pp. 596–599, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[21] Ashish John, Ajay Agarwal, and Manish Bhardwaj, "An Adaptive Algorithm to Prevent SQL Injection," *American Journal of Networks and Communications*, vol. 4, no. 3-1, pp. 12–15, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[22] Chen Ping et al., "Research and Implementation of SQL Injection Prevention Method based on ISR," *IEEE International Conference on Computer and Communications*, pp. 1153–1156, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[23] G. Buja, K. B. Abd Jalil, et al., "Detection model for SQL injection attack: an approach for preventing a web application from the SQL injection attack," Symposium on Computer Applications and Industrial Electronics, IEEE, pp. 60–64, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[24] Ahmad Ghafarian, "A Hybrid Method for Detection and Prevention of SQL Injection Attacks," *IEEE Computing Conference*, pp. 833–838, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[25] Nilesh Yadav, and Narendra Shekokar, "SQLI Detection Based on LDA Topic Model," *International Journal of Engineering Trends and Technology*, vol. 69, no. 11, pp. 47-52, 2021. [CrossRef] [Publisher Link]

[26] M. Amutha Prabakar, M. Karthikeyan, K. Marimuthu, "An Efficient Technique for Preventing SQL Injection Attack Using Pattern Matching Algorithm," *IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology*, pp. 503–506, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[27] Debasish Das, Utpal Sharma, and D.K. Bhattacharyya, "An Approach to Detection of SQL Injection Vulnerabilities Based on Dynamic Query Matching," *International Journal of Computer Applications*, vol. 1, no. 25, pp. 28-34, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[28] Mukesh Kumar Gupta, Mahesh Chand Govil, and Girdhari Singh, "A Context-Sensitive Approach for Precise Detection of Cross-Site Scripting Vulnerabilities," *International Conference on Innovations in Information Technology (IIT)*, pp. 7-12, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[29] Ashwin Ramesh, Anirban Bhowmick, and Anand Vardhan Lal et al., "An Authentication Mechanism to Prevent SQL Injection by Syntactic Analysis," *International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15)*, pp. 1–6, 2015. [CrossRef] [Google Scholar] [Publisher Link]