IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

SQL Injection Attacks and Prevention Techniques

[1]Maharudra Ganjure, [2]Vishal Sarode, [3]Pallavi Chavan

[1]Student, [2]Student, [3]Professor

[1]Department of Artificial Intelligence and Data Science,

[1]Indira College of Engineering and Management, Pune, India

Abstract: Among the most threatening security flaws for web applications is SQL injection (SQLi), a kind of attack that seeks to take advantage of vulnerabilities in database queries to access unauthorized information. This paper explores all types of SQL injection attack, including classic, blind and error based sqli by giving the detailed example how this vulnerability impacts integrity of database. Technologies — input validation, parameterized queries, Web Application Firewalls (WAFs) are detailed as formidable defenses against SQLi attacks. Web applications can use these mitigation strategies to protect against data breaches and unauthorized access. It also highlights the need to implement secure coding practices and adopt encryption for data at rest building a defense in depth solution. The paper extra spotlights real stories to present the item additionally and exactly how well sure prevention techniques can shrink the SQLi attack surface.

Keywords - SQL Injection, Web Security, Parameterized Queries, Input Validation, Web Application Firewalls, Database Security, Prepared Statements.

I. INTRODUCTION

SQL injection (SQLi) is considered to be one of the top web application vulnerabilities which allow attackers to control an app and access its data by affecting queries made between an app, client-side interface, and backend DBM. Attackers also exploit the mistakes of SQL queries, unauthorized access to confidential data, modification of records, or deleting important information. Cross site scripting attacks can be very dangerous for web applications, particularly those that have an important amount of data on database side of users including usernames, passwords, financial data and personal information.

SQLi attacks can be devastating and are quite simple to carry out, often resulting in data breaches, identity theft, and loss of reputation. More conventional techniques have evolved over the years to prevent these attacks, e.g., input validation and parameterized queries, as well as more advanced ones like WAFs (Web Application Firewalls). While these strategies help to mitigate the issue, SQLi continues to riddle applications because of bad coding practices and evolving attack vectors.

In the present paper, different ways in which SQLi might be performed such as classic, blind and error based shall be explored followed by a discussion on the newest methods used to prevent it. If developers can know how SQL injection works and apply best security practices, databases will always be safe from unauthorized access and the overall system remains intact.

II. LITERATURE REVIEW

SQL injection (SQLi) is one of the main issues in cyber security research for many years and it has been extensively used to exploit web application weaknesses. This paper so far presented the various papers which address techniques using different algorithms to detect and prevent these attacks. In the early days a lot of work was done to apply preventive measures, mainly input validation and parameterized queries. For instance, Halfond et al. While the former (Chen et al., 2006) even stated that input validation is first line of defense, this approach sometimes fail when it is not implemented in all application layers consistently.

Follow-up work, however, has reviewed more rigorous techniques. Kapravelos et al. Only web application firewalls (WAFs) as an extra screen have been existed which should filter most of the malicious traffic o more efficiently way before reaching the application. On the other hand, as WAFs are predicated on adapting to new attack vectors, they have limits against attackers who use advanced evasion techniques. Work like Bisht & Venkatakrishnan (2015) have demonstrated dynamic analysis tools that monitor the behavior of web applications at runtime to detect potential SQLi attempts but these techniques are often heavyweight and not always suitable for large scale applications.

An additional challenge that has appeared to receive recent focus in academia is the mitigation of more sophisticated types of SQLi — blind and error-based injections. There are a large volume of resources about traditional SQLi exploits, but blind SQLi — which reveals no error messages to attackers — presents a challenge that is less well documented. For instance, Shahriar and Zulkernine (2017) pointed out how hard it is to detect these stealth attacks because they are not detected by conventional security mechanisms.

While there have been overall improvements, little research has been conducted on SQLi properties and the newly developed methods of attack: timing based and out-of-band SQLi. A lot of research also has suggested different tools to automate the detection of these vulnerabilities, but there is still a gap in getting full-fledged solutions that can evolve with new attack vectors. However, significant challenges remain: as web applications continue to evolve and grow in complexity, we need to explore the development of scalable, lightweight defense mechanisms that can efficiently result in zero missed detection Model FPs across multiple deployment environments.

III. METHODOLOGY

The current research accounts for three significant categories of attacks on SQL injection. These attacks include classic SQL injection, blind SQL injection, and error-based SQL injection. All these attacks manipulate the database operations as well as extract sensitive information based on different vulnerabilities of web application query handling.

1. Classic SQL Injection

It is a classic SQL injection where the attacker injects malicious SQL code through input fields, for instance, forms, URLs, or cookies. In most cases, unauthorized extraction or manipulation of data occurs. In this research, the usual approaches to classic SQL injection were used to experiment with known vulnerable web applications, which led to this analysis on common security flaws.

2. Blind SQL Injection

Blind SQL injection does not produce visible error messages, making it hard to discover. Instead, the attackers have to only assume whether the SQL query is successful or not by relying on the changes in the behavioral application. For this research, both time-based and Boolean-based tests are performed using blind SQL injection. These applications being tested deliberately are as vulnerable as those in real-time attacks done by malicious users to bypass silently through security checks.

3. Error-Based SQL Injection

Error-based SQL injection is based on the principle that uses error messages from the database for gathering information. This is done by such a design in SQL queries that the application will throw some error disclosing valuable information. The vulnerability assessment tools SQLMap were implemented in an attempt to simulate this type of attack to identify vulnerabilities.

Tools and Techniques

Using a combination of penetration testing and vulnerability assessment tools: SQLMap is an advanced automated SQL injection tool that was used for simulating attacks and for evaluating preventive measures at both code and configuration levels. In addition to SQLMap, several other web vulnerability scanners, like OWASP ZAP, were used in the testing phase to probe for weaknesses in the source code of the tested

applications. Data extracted from these tools was then analyzed to determine the attack surface with respect to different types of SQLi vectors and resulting risk levels.

Data Collection

The experiment was performed with the deployment of testing environments with real-world applications and purposes using vulnerable platforms: DVWA (Damn Vulnerable Web Application) and Mutillidae. These presented controlled conditions where the author could conduct experiments with SQL injection and test the outcome. Data collected while penetrating was recorded and analyzed to observe the success of differing security controls, including input
validation, prepared statements, and parameterized queries.

How SQL Injection Works?

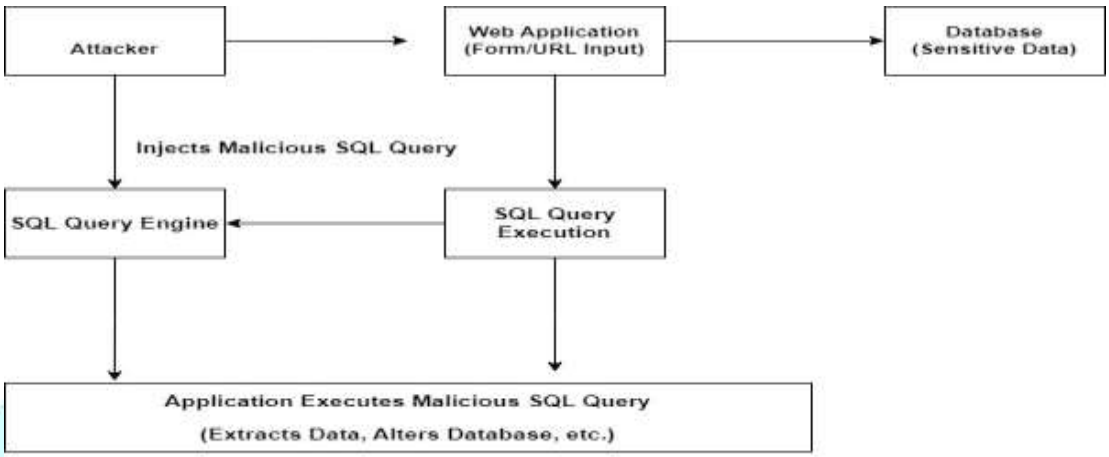

Figure 1: Data Flow Diagram for SQL Injection attack process

SQL Injection is one of the most common and dangerous vulnerabilities within web applications. SQLi achieves this by injecting malicious SQL statements into weaknesses in the handling of user input in SQL queries to gain access, manipulate databases, or retrieve sensitive information. This chapter describes the SQLi mechanism based on its two common forms: Union-based SQLi and Boolean-based SQLi.

How a SQL Injection Attack Works?

A SQL Injection attack can be very simple: just insert or "inject" a SQL query as part of the input data from the client to the application. When a successful attack is triggered, it allows the attacker manipulation over the SQL queries that an application would use in its interaction with its database. It usually consists of the following phases:

Vulnerable inputs**:** The attacker identifies the vulnerabilities of those inputs inside the web application. It could be a simple text field inside a form, a URL parameter, or some sort of other input device.

Malicious SQL Statement Creation: Here an attacker uses a vulnerable input to create an SQL statement that will be inserted into the query run by the application. In this, such a statement is crafted to alter the original SQL statement so it carries out actions not intended by the application developers.

Bypassing application security measures: The attackers bypass security measures by either input validation or special character escaping. The attackers use this to their advantage by utilizing techniques, including string concatenation or SQL syntax that will comment out parts of the original query. In the execution of the malicious query, the SQL query will include the malicious input of the attacker. This altered query can function to perform unauthorized access of the data, deletion of data, or even alteration of the database schema.

The outcome could be that sensitive information like user credentials may be extracted, existing data modified, new data may be added, or a significant amount of the database deleted. This relies on the attack. The vulnerabilities of the database server: Advanced SQL injections can exploit the weaknesses within the database server, and the attack may reach beyond the database to attack the server level by executing operating system commands or by trying to access other parts of the file system maintained by the server. In a nutshell,

such process relies on dynamic SQL execution at those places in the application where user input is embedded in the SQL statements without proper validation or escaping. Its method of attack exploits how SQL queries are constructed, which developers often do not anticipate.

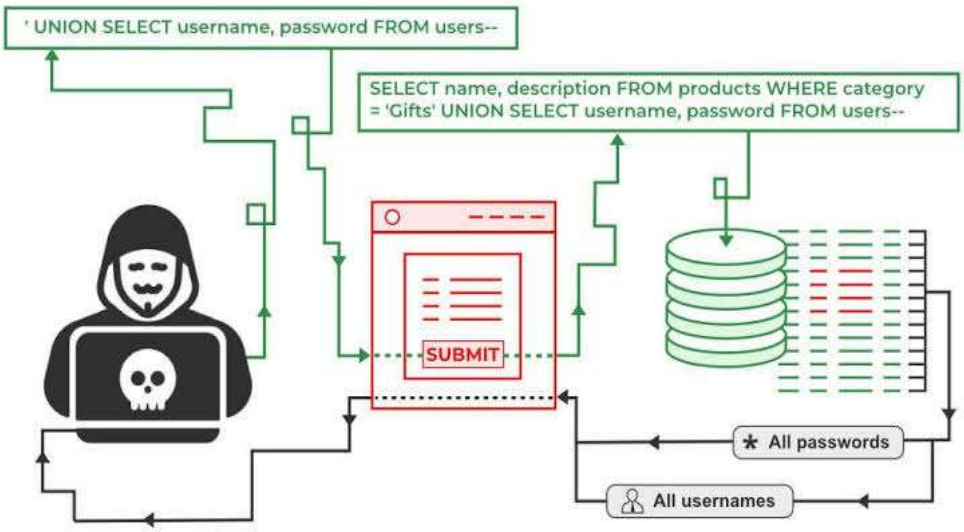

Figure 2: Union-Based SQL Injection

The second diagram is the Union-based SQL Injection attack. This is the type of attack which utilizes the UNION SQL operator so as to add the queries alongside the existing SQL query to retrieve sensitive data from the database.

Explanation:

The attacker injects a malicious SQL payload such as UNION SELECT username, password FROM users--, in a vulnerable form of a web application.

The Web application accepts this input and appends it to the existing SQL query, like this: SELECT name, description FROM products WHERE category = 'Gifts' UNION SELECT username, password FROM users--.

Thus, the malicious query fetches not only product-related data but also usernames and passwords from the "users" table.

Therefore, all usernames and passwords are sent back to the attacker, in turn leading to a complete system compromise. The diagram clearly shows how the attacker could manipulate query results or even acquire unauthorized access to information through insecure input handling.

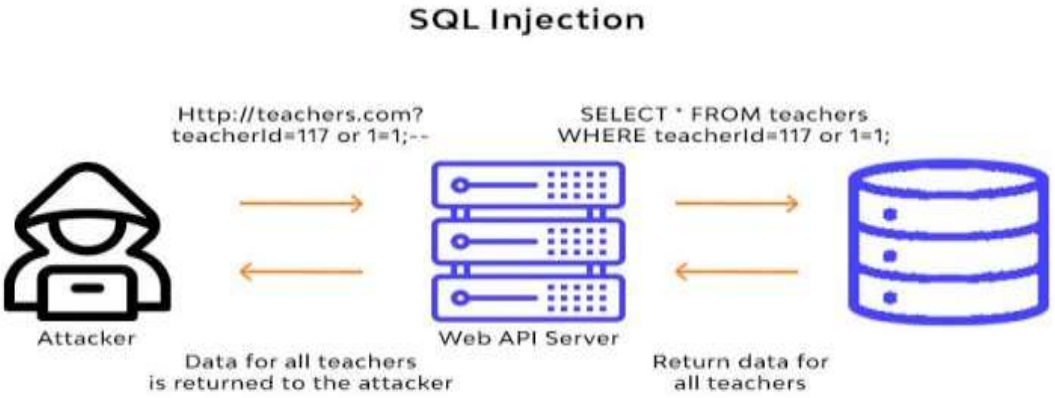

Figure 3: Boolean-Based SQL Injection

As said earlier, the third diagram illustrates a Boolean-Based SQL Injection. It is actually another form of blind SQL injection in which the application responds differently for varying scenarios depending upon the factuality of a query. This technique is good when you cannot get direct data but can infer information from applications' behaviors.

Explanation:

The attacker sends a query with a conditional payload, e.g., SELECT * FROM teachers WHERE teacherId=117 OR 1=1;--.

In this example, the condition 1=1 is always true and the database will return all records of the "teachers" table.

The server then continues to execute the query and returns data from all the teachers to an attacker, although this query should return data from only one teacherId.

This attack allows the attacker to conclude the structure of the database, and extract information merely by testing conditions.

This graphic illustrates how, by searching for the lack of specific logical conditions in SQL queries, an attacker can extract information without ever having a view of the results of a single query. This is a common technique used when the attacker obtains no form of extended error messages, or rendered data output.

Prevention Techniques

SQL Injection actually manifests itself when an attacker tries to inject malicious SQL code into the database that is used by a web application. It becomes one of the critical security concerns that could have quite evil consequences, including data theft, manipulation of data, and even complete system compromise. Nevertheless, various defense mechanisms have been devised to effectively mitigate the risks associated with SQL injection attacks. Some of the most widely accepted methods are input validation, parameterized queries, stored procedures, web application firewalls (WAFs), and the principle of least privilege. Each one of these techniques plays a very important role in securing web applications and databases.

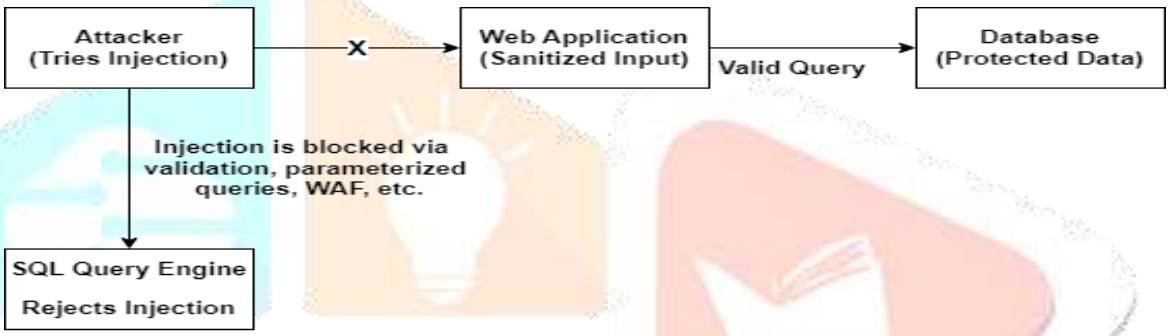

Figure 4: SQL Injection Prevention

Key Points in the DFD:

Attacker: He uses SQL injection with the input fields of the web application.

Web Application: It validates, parameterizes queries, and throws the malicious input.

Database: This database, with strong coding defense practices, has been so safeguarded that only valid queries execute.

1. Input Validation: Input validation is the verification that user-supplied data is clean and in line with expected formats before processing through the application. Checking for specific types of inputs, such as rejecting special characters like quotes, semicolons, or SQL-specific terms, by and large can significantly minimize chances for successful SQL injection attacks.

Example: A numeric field like ID, will need validation so that only the numeric values will go through and reject any malicious string like OR 1=1.

2. Parameterized Queries and Prepared Statements

SQL injection can actually be prevented using parameterized queries or prepared statements-those techniques help ensure that a given SQL code does not get mixed with user input at any point, ensuring that the input cannot be interpreted as executable SQL code.

It works because it is not injecting user inputs directly into SQL queries; instead, it creates a template of a SQL query with placeholders in it (such as ? or @parameter) for which actual data is bound at runtime. In this situation, the user's input has no chance to alter the structure of the query because it is not part of the SQL query itself.

Example:

```
// Vulnerable query
String query = "SELECT * FROM users WHERE username = '" + userInput + "' AND password = '" +
password + "'";

// Safe query using parameterized statement
PreparedStatement pstmt = connection.prepareStatement("SELECT * FROM users WHERE username = ?
AND password = ?");
pstmt.setString(1, userInput);
pstmt.setString(2, password);
ResultSet rs = pstmt.executeQuery();
```

The other type of protection against SQL injections is the separation between queries and data. Hence, SQL injections such as userInput = ' OR 1=1 -- ' would not work since their input is strictly taken as data and not as any SQL code.

3. Stored Procedures

Stored procedures can also prevent SQL injection attacks by encapsulating the SQL logic inside the database. Procedures are parameter-driven, which can be built with less chance of wrong SQL syntax execution. So, stored procedures again place another layer of abstraction in the way a user input goes before it gets executed.

Example:

```
CREATE PROCEDURE getUserData
  @username NVARCHAR(50),
  @password NVARCHAR(50)
AS
BEGIN
  SELECT * FROM users WHERE username = @username AND password = @password;
END;
```

Stored procedures help prevent direct injection of SQL code from being placed directly into queries, if the procedure is used appropriately.

4. Web Application Firewalls (WAFs)

A Web Application Firewall, or WAF, filters and inspects incoming traffic to a web application. It can readily detect and prevent SQL injection attempts by looking into incoming HTTP requests for known patterns of malicious activity. While no substitute in any way for proper input validation and the use of prepared statements, a WAF does afford an additional line of defense against specific automated SQL injection attacks. How it works: WAFs will detect anomalous or suspicious requests, especially those containing SQL keywords such as SELECT, INSERT, or UNION, and are deviating from normal traffic patterns. They can therefore block such requests before they get to the server, thus reducing the attack based on SQL injection.

5. Least Privilege Principle

The principle of least privilege is followed by giving as few privileges as possible to the database accounts utilized by the application. For example, if an application only needs read access to a database, then user accounts interacting with a database must not have privileges that are authorized for writing or modifying data.

How does it work?

It limits the capabilities of the database account. Even if the attacker successfully injects SQL code, their ability to manipulate the database will be limited. This reduces the damage from a successful attack to the extent possible.

SQL injection prevention example

Login page with SQL-injection vulnerability

SELECT * FROM users WHERE username = 'admin' AND password = '' OR 1=1 – '

There is a chance that an attacker could log in as an admin without a password because the SQL query is manipulated.

Prevention: Using parameterized queries:

SELECT * FROM users WHERE username = ? AND password = ?

This would ensure that the input by an attacker is not interpreted as part of the SQL query because input is always interpreted as data.

Case Study / Real-World Examples

SQL injection attacks are something that has been looming around the security sphere for decades, and in many cases, bring about devastating breaches of data and total system compromise. The following are real-life examples of SQL injection attacks that depict the severity of such vulnerabilities and how proper prevention techniques could have mitigated such risks.

Real-Life SQL Injection Attack Examples

1. GhostShell Attack on Universities

In 2012, Team GhostShell, a hacking group affiliated with the APT group, launched a spectacular SQL injection attack against 53 universities worldwide. The attackers used vulnerable input fields on university websites to inject malicious SQL queries. This resulted in their getting over 36,000 personal records of students, faculty, and staff posted online. This vulnerability reflected the lack of input validation and highlighted the importance of proper query parameterization in the universities.

Avoidance: The attack could have avoided if the web application universities have taken measures such as input validation, parameterized queries, and WAFs in place to recognize and block SQL injection attempts.

2 .RedHack Collective's Cyber-Attack on the Turkish Government

Another APT group, which was referred to as RedHack, used SQL injection vulnerabilities in government Turkish websites. In a targeted attack, RedHack used SQL injection in order to compromise the database and delete state debt records from state agencies' record, causing significant financial disruption.

Prevention: It would have been possible by use of stored procedures and the principle of least privilege; because the compromised web application only had limited access to the most vital systems.

3.7-Eleven Breach

Perhaps the largest and most memorable SQL injection attacks were those launched in 2007, when a group of hackers broke into the systems of several companies. One of the most notable victims at that time was the 7-Eleven retail chain. Using SQL injection attacks, these attackers stole over 130 million credit card numbers. Such an attack seriously compromised the web application architecture of the company and the approach to processing user input.

Prevention: The attackers could not inject malicious SQL code in the backend of the web application through parameterized queries and input validation, thereby preventing theft of credit card information.

Significant SQL Injection Vulnerabilities

Tesla Vulnerability (2014)

Tesla researchers in 2014 discovered an SQL injection vulnerability in the site of Tesla. This had made attackers inject a malicious query to gain full access to the database with administrative access, which could threaten the sensitive information of the users involved. Luckily, that vulnerability was reported by the **researchers prior to its major breach occurrence.**

Prevention: It would not have been so likely to have such a SQL injection vulnerability had a least privilege model and prepared statements been used, since the input could not alter the original SQL query structure.

Cisco Vulnerability (2018)

A serious SQL injection flaw was found in Cisco Prime License Manager, which could be exploited to get the shell access to the systems on which license managers were present. This paved the way to further attacks that were even more malicious. Cisco issued a patch to remediate this flaw with great promptness. As this suggests, even companies of illustrious status can be prone to vulnerabilities in their systems.

Prevention: Had Cisco employed input sanitization along with parameterized queries, it wouldn't have permitted attackers to inject the malicious SQL code, thus preventing them from obtaining shell access in their systems.

Discussion and Future Scope

Input validation, parameterized queries, prepared statements, and web application firewalls are some of the defense mechanisms that exist to prevent SQL injection attacks. Input validation can be relied upon to prevent most common SQL injection attacks. Though this mechanism can block most SQL injection attacks by treating input as data rather than as a part of the SQL query, the success of parameterized queries and prepared statements has been much more significant in this regard.

The fact, though, is that SQL injection techniques are evolving with attackers discovering novel approaches like blind SQL injection and time-based SQL injection to bypass traditional defenses. These new approaches decisively point toward the need for continuing improvement in prevention methods.

Future Research Directions

AI-Based Detection Systems: Develop systems using machine learning algorithms that can detect abnormal patterns in real-time, thus offering a chance for the automatic detection and response of threats.

Advanced Encryption Techniques: Homomorphic encryption, through which computation can be made securely over the encrypted data without decrypting it, to limit the damage SQL injection can do.

Blockchain: Integration of blockchain technology into the maintenance of a distributed ledger of all the database transactions. It would go a long way in identifying and preventing unauthorized changes.

Automated Security Scans: Incorporation of automated scanning tools that constantly scan application code and configurations of databases for vulnerabilities so that one remains secure against SQL injection attacks.

IV. CONCLUSION

SQL injection attacks are part of many web application security threats, which have the potential to cause serious data breaches that may badly destroy organizations. This paper especially focused on highlighting the significance of securing databases through an analysis of all kinds of attack vectors, real incidences that occurred in the real world, and efficiency rates of different prevention techniques, including input validation, parameterized queries, stored procedures, and web application firewalls (WAFs). Although these techniques form a solid base for defense, advanced attacks require ceaseless innovation and vigilance. Cases such as GhostShell and 7-Eleven attack situations showcase catastrophic potential of the possible SQL injection vulnerabilities and demonstrate the importance of active security. Organizations can reduce these threats only if they implement tight input validation, parameterized queries, and frequent deployments of WAFs with the principle of least privilege. Prevention of SQL injection will then require multiple levels-approaches: best coding practices and the newer security technologies-for reducing the risks.

V. REFERENCES

[1] K. Gaur, M. Diwakar, K. Gaur, P. Singh, T. Sachdeva, and N. K. Pandey, "SQL Injection Attacks and Prevention," in 2023

6th International Conference on Information Systems and Computer Networks (ISCON), Mathura, India, 2023, pp. 1-4,

doi: 10.1109/ISCON57294.2023.10112156.

[2] A. Rai, M. M. I. Miraz, D. Das, H. Kaur, and Swati, "SQL Injection: Classification and Prevention," in 2021 2nd International

Conference on Intelligent Engineering and Management (ICIEM), London, United Kingdom, 2021, pp. 367-372, doi:

10.1109/ICIEM51511.2021.9445347.

[3] P. Kumar and R. K. Pateriya, "A Survey on SQL Injection Attacks, Detection and Prevention Techniques," in 2012 Third

International Conference on Computing, Communication and Networking Technologies (ICCCNT'12), Coimbatore, India,

2012, pp. 1-5, doi: 10.1109/ICCCNT.2012.6396096.

[4] H. Gupta, S. Mondal, S. Ray, B. Giri, R. Majumdar, and V. P. Mishra, "Impact of SQL Injection in Database Security," in

2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, United Arab

Emirates, 2019, pp. 296-299, doi: 10.1109/ICCIKE47802.2019.9004430.

[5] Li Qian, Zhenyuan Zhu, Jun Hu, and Shuying Liu, "Research of SQL Injection Attack and Prevention Technology," in 2015

International Conference on Estimation, Detection and Information Fusion (ICEDIF), Harbin, China, 2015, pp. 303-306, doi:

10.1109/ICEDIF.2015.7280212.

[6] Yong Fang, Jiayi Peng, Liang Liu and Cheng Huang, "WOVSQLI: Detection of SQL injection behaviors using word vector

and LSTM", Proceedings of the 2nd international conference on cryptography security and privacy, pp. 170-174, 2018.

[7].V. Anitha, Supha A. Lakshmi, M. Revathi and K. Selvi, "Detecting various SQL Injection vulnerabilities using String

Matching and LCS method", 2014 Sixth International Conference on Advanced Computing (ICoAC), pp. 237-241, 2014.