International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 4 Volume No.9 July - August – 2025
DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

Mitigating SQL Injection Attacks: A Comprehensive Study on Attack Vectors, Classification, and Prevention Techniques

Rajesh Tanti
Department of Computer Science and Engineering , OP Jindal University, Raigarh (C.G.)

Abstract
SQL Injection (SQLi) is a severe and widely exploited security vulnerability in web applications that enables attackers to manipulate backend databases through maliciously crafted input. This attack compromises the confidentiality, integrity, and availability of data by allowing unauthorized access, data leakage, and even complete system takeover. SQL Injection remains one of the top threats listed in the OWASP Top 10 due to persistent flaws in input validation and query handling.
This paper provides a detailed classification of SQL Injection attacks, including Classic SQL Injection, Blind SQL Injection (Boolean-based and Time-based), Error-based SQL Injection, and Out-of-Band SQL Injection. Each type exploits different system behaviors and levels of database exposure to achieve malicious goals.
In response to these threats, the paper explores multiple prevention techniques such as parameterized queries (prepared statements), stored procedures, input validation and sanitization, the principle of least privilege, and deployment of Web Application Firewalls (WAF). It also emphasizes the importance of secure development practices, regular code audits, and adherence to security standards like those provided by OWASP.
Keywords: SQL Injection, Web Application Security, Input Validation, Prepared Statements, Blind SQLi, Error-based SQLi, Out-of-Band SQLi, Cybersecurity, OWASP, Secure Coding, Data Protection, WAF, Database Security.

Introduction :

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an
application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behaviour.
In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks.The Figure 1. Showing that a demo picture which describe how the attacker try to connect with the target and perform the sql attack .

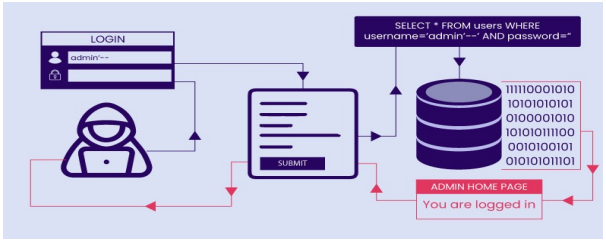

Figure 1 : General work flow diagram for SQL injection attack

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 4 Volume No.9 July - August – 2025
DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

SQL injection (SQLi) is a cyberattack that injects malicious SQL code into an application, allowing the attacker to view or modify a database. According to the Open Web Application Security Project, injection attacks, which include SQL injections, were the third most serious web application security risk in 2021. In the applications they tested, there were 274,000 occurrences of injection.

To protect against SQL injection attacks, it is essential to understand what their impact is and how they happen

so you can follow best practices, test for vulnerabilities, and consider investing in software that actively prevents attacks.

Case Study :

1. Heartland Payment Systems (2008)

One of the largest payment processors in the U.S. suffered a massive SQL Injection breach, compromising approximately 130 million credit and debit card numbers. The incident highlighted severe vulnerabilities in financial infrastructure and acted as a pivotal moment in payment system security awareness.

2. Sony Pictures (2011)

A SQL Injection attack on Sony's network compromised 77 million PlayStation Network accounts, leading to significant data loss and an estimated $170 million in damages. This breach emphasized the susceptibility of even large corporations with advanced digital infrastructure.

3. Yahoo! Voices (2012)

In a major attack affecting Yahoo!'s content platform, hackers exploited SQL Injection to leak around 500,000 email addresses and passwords. The breach exposed critical weaknesses in web application security and user data protection.

4. TalkTalk (2015)

UK telecom provider TalkTalk experienced a cyberattack that exposed the personal data of approximately 157,000 customers. The incident served as a wake-up call for the telecommunications industry regarding the need for robust and proactive cybersecurity strategies.

5. Capital One (2019)

A misconfigured web application firewall enabled a SQL Injection exploit that led to the breach of personal data belonging to over 100 million customers, including names, addresses, and credit scores. This case underscored the risks of insecure cloud configurations and web services.

6. High-Profile Retailer Breach (2025)

In 2025, a major retailer was breached via SQL Injection, exposing customer credit card and personal information. The attack resulted in significant financial losses and reputational damage, demonstrating that basic vulnerabilities remain a threat even in modern systems.

7. Financial Institution Breach (2025)

Also in 2025, attackers exploited a SQL Injection vulnerability in a financial institution to manipulate transaction data, impacting the integrity of financial records. The breach triggered regulatory fines and raised concerns about transactional security in the financial sector. Figure 2 we can seen that the SQL attack is started from 1998 to till now , attacker the powerful techniques to find vulnerability and attack on target

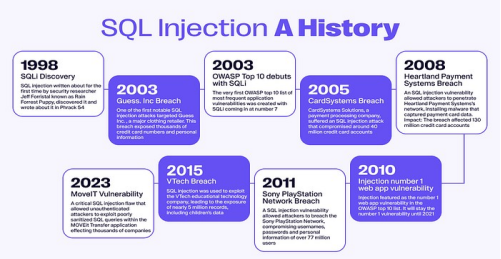

Figure 2: Historical attack scenario

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 4 Volume No.9 July - August – 2025
DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

Following figure 3 showing the increasing attack scenario of SQL injection attack year wise, by the analysis of the diagram we found SQL injection attack increasing on average 25% in each year .

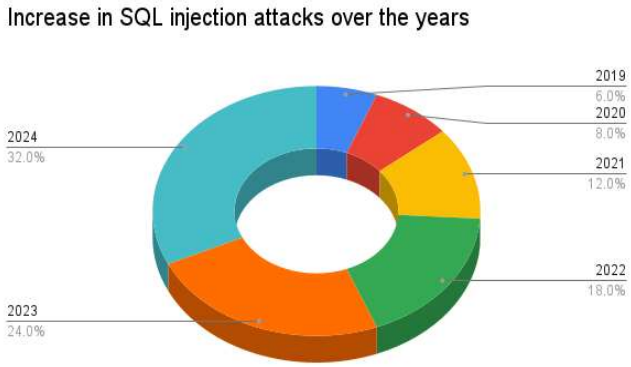

Figure 3: SQL attack scenario on year wise percentile.

Consequences of a Successful SQL Injection Attack

SQL injection attacks can have a significant negative impact on an organization. Organizations have access to sensitive company data and private customer information, and SQL injection attacks often target that confidential information. When a malicious user successfully completes an SQL injection attack, it can have any of the following impacts:

Exposes Sensitive Company Data: Using SQL injection, attackers can retrieve and alter data, which risks exposing sensitive company data stored on the SQL server.

Compromise Users' Privacy: Depending on the data stored on the SQL server, an attack can expose private user data, such as credit card numbers.

Give an attacker administrative access to your system: If a database user has administrative privileges, an attacker can gain access to the system using malicious code. To protect against this kind of vulnerability, create a database user with the least possible privileges.

Give an Attacker General Access to Your System: If you use weak SQL commands to check usernames and passwords, an attacker could gain access to your system without knowing a user's credentials. With general access to your system, an attacker can cause additional damage accessing and manipulating sensitive information.

Compromise the Integrity of Your Data: Using SQL injection, attackers can make changes to or delete information from your system.

Threat Modeling

SQL injection attacks allow attackers to spoof identity, 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, or become administrators of the database server.

SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Due to the nature of programmatic interfaces available, J2EE and ASP.NET applications are less likely to have easily exploited SQL injections.

The severity of SQL Injection attacks is limited by the attacker's skill and imagination, and to a lesser extent, defense in depth countermeasures, such as low privilege connections to the database server and so on. In general, consider SQL Injection a high impact severity.

SQL injection attack occurs when:

An unintended data enters a program from an untrusted source.

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 4 Volume No.9 July - August – 2025
DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

The data is used to dynamically construct a SQL query
The main consequences are:
Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL Injection vulnerabilities.
Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.
Authorization: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL Injection vulnerability.
Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection attack.
Impact of sql injection attack
Unauthorized access to sensitive data: Attackers can retrieve personal, financial, or confidential information stored in the database.
Data integrity issues: Attackers can modify, delete, or corrupt critical data, impacting the application's functionality.
Privilege escalation: Attackers can bypass authentication mechanisms and gain administrative privileges.
Service downtime: SQL injection can overload the server, causing performance degradation or system crashes.
Reputation damage: A successful attack can severely harm the reputation of an organization, leading to a loss of customer trust.

Risk Factors
The platform affected can be:
Language: SQL
Platform: Any (requires interaction with a SQL database)
SQL Injection has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind.
Essentially, the attack is accomplished by placing a meta character into data input to then place SQL commands in the control plane, which did not exist there before. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.
Examples
In SQL: select id, firstname, lastname from authors
If one provided: Firstname: evil'ex and Lastname: Newman
the query string becomes:
select id, firstname, lastname from authors where firstname = 'evil'ex' and lastname ='newman'
which the database attempts to run as:
Incorrect syntax near il' as the database tried to execute evil.
A safe version of the above SQL statement could be coded in Java as:

```
String firstname = req.getParameter("firstname");
String lastname = req.getParameter("lastname");
// FIXME: do your own validation to detect attacks
String query = "SELECT id, firstname, lastname FROM authors WHERE firstname = ? and lastname = ?";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, firstname );
pstmt.setString( 2, lastname );
try
{
   ResultSet results = pstmt.execute( );
```

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 4 Volume No.9 July - August – 2025
DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

}

Procedure Used in a Database Attack

1. Identification:
The attacker begins by identifying a potential target system. This often involves scanning techniques such as port scanning or network reconnaissance to locate accessible systems and services.

2. Information Gathering:
Once the target is identified, the attacker collects detailed information about it. This may include operating system details, database type and version, open ports, user accounts, and configurations.

3. Network Mapping:
The attacker maps the network to understand its topology and structure. This helps identify how the database is connected, what other systems it communicates with, and potential entry points.

4. Finding Vulnerabilities:
The attacker analyzes the target for weaknesses such as outdated software, misconfigurations, weak authentication mechanisms, or known exploits related to the database system.

5. Exploitation:
After identifying vulnerabilities, the attacker exploits them to gain unauthorized access to the database. This could involve SQL injection, privilege escalation, or exploiting software bugs.

6. Finishing / Covering Tracks:
Once the attack is complete, the attacker may try to cover their tracks by deleting logs, creating backdoors for future access, or exfiltrating data without detection.

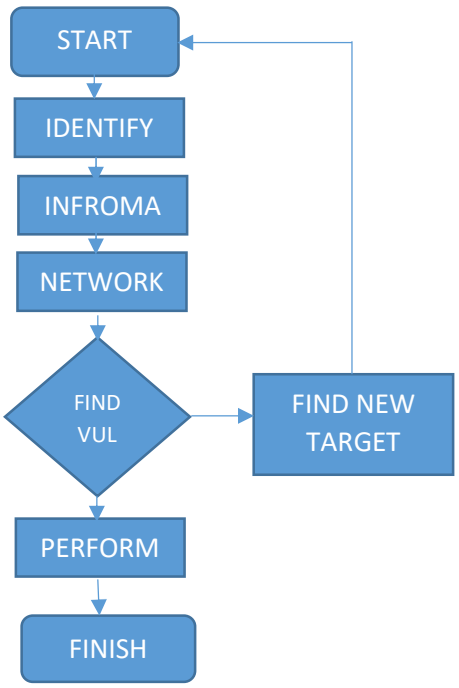

Figure 4: Flow chart SQL attack procedure

Litrature Review:

1. **"Mitigation of SQL Injection Attacks Through Machine Learning Classifier"** by Dr. P. Anu et al. (10 July 2024) analyzes SQL injection attack vectors and reviews existing literature while proposing machine learning-based detection methods. The study utilizes classifiers like KNN,

DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

Random Forest, Logistic Regression, Perceptron with SGD, and Voting Classifier to identify SQL injection attacks effectively. The paper highlights the practical value of enhancing web database security through these models. However, it notes that evolving attack vectors remain a challenge and that many existing mitigation methods have become outdated.

2. **"Understanding the Threat: Exploring SQL Injection Attacks and Prevention Strategies"** (31 May 2024, IRJMET&S) focuses on SQL injection methodologies, real-world cases, and impacts, offering a broad overview of how attackers exploit web application vulnerabilities. The paper stresses the need for proactive prevention techniques to secure systems and emphasizes a sector-wide understanding of attack vectors. While insightful, it does not identify specific research gaps or limitations.

3. **"A Diligent Survey of SQL Injection Attacks"** by Nilima D. Bobade et al. (24 February 2024) categorizes SQLi into classical and advanced types and evaluates both detection and prevention methods. It contributes significantly by comparing techniques and incorporating deep learning discussions, offering a strong foundation for addressing database-targeted cybersecurity threats. No specific limitations or research gaps are identified in the paper.

4. **"SQL Injection and Prevention"** by S. Yaswanthraj et al. (1 June 2024, IJRPR) discusses SQLi as a common attack vector and outlines various preventive measures like input validation, stored procedures, and web application firewalls. It explains how these mechanisms help avoid data breaches and financial loss. However, the study lacks detailed insights into research gaps or limitations.

5. **"Beyond the Basics: A Study of Advanced Techniques for Detecting and Preventing SQL Injection Attacks"** by Ashtha Goyal and Priya Matta (20 September 2023) explores both traditional and advanced methods, including runtime monitoring, taint analysis, machine learning-based anomaly detection, and behavior-based techniques. It aims to bolster application stability and data protection but does not elaborate on specific gaps or limitations.

6. **"A Detailed Evaluation of SQL Injection Attacks, Detection and Prevention Techniques"** by Nikita Joshi et al. (2 December 2022) reviews 38 scholarly publications to analyze SQLi attack vectors and prevention strategies. The paper offers a solid framework for developing effective SQLi defenses but is limited in scope due to the narrow set of literature and limited coverage of preventive methods.

7. **"Diverse Approaches Have Been Presented to Mitigate SQL Injection Attack, But It Is Still Alive: A Review"** by Mohammad Qbea'h et al. (20 December 2022) critiques existing mitigation models, pointing out their weaknesses and calling for innovative, comprehensive solutions. It employs static, dynamic, hybrid, and machine learning approaches to underline the persistent challenges in SQLi defense. The study notes that current methods often fail to address SQLi thoroughly and require more robust, complete strategies.

8. **Another version of "A Detailed Evaluation of SQL Injection Attacks, Detection and Prevention Techniques"** (also dated 2 December 2022) reiterates insights from paper #6, offering strategies for modifying SQL queries to prevent unauthorized access. It highlights foundational knowledge but lacks comprehensive evaluation metrics and has a limited focus on prevention techniques.

9. **"A Comprehensive Study on SQL Injection Attacks, Their Mode, Detection and Prevention"** by Sabyasachi Dasmohapatra et al. (1 January 2022) covers types of SQLi, how attackers exploit them, and prevention strategies including novel algorithms for encrypting data queries. It aims to reduce the impact of such attacks on database systems but does not identify research gaps or limitations.

10. **"A Classification of SQL-Injection Attacks and Countermeasures"** by William Halfond, Jeremy Viegas, and Alessandro Orso (1 January 2006) presents an early but thorough classification of SQLi attack types and countermeasures. It evaluates the strengths and weaknesses of each prevention method and emphasizes that many techniques do not cover the full range of SQLi threats.

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 4 Volume No.9 July - August – 2025
DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

Limitations include reduced effectiveness against emerging or hybrid attacks and over-reliance on known patterns.

How Does SQL Injection Work?

SQL Injection typically works when a web application improperly validates user input, allowing an attacker to inject malicious SQL code. For example, if a web application takes user input (e.g., a username or password) and directly inserts it into an SQL query without proper sanitization, an attacker can manipulate the query to perform unintended actions.

Example:

Suppose we have an application based on student records. Any student can view only his or her records by entering a unique and private student ID.

SELECT * FROM STUDENT WHERE STUDENT_ID == 12222345 or 1 = 1

SQL Injection based on 1=1 is always true. As we can see in the above example, 1=1 will return all records for which this holds true. So basically, all the student data is compromised. Now the malicious user can also similarly use other SQL queries.

Query 1:

SELECT * FROM USER WHERE USERNAME = “” AND PASSWORD=””

Now the malicious attacker can use the ‘=’ operator to retrieve private and secure user information. So following query when executed retrieves protected data, not intended to be shown to users.

Query 2:

SELECT* FROM User WHERE (Username = “” OR 1=1) AND (Password=”” OR 1=1).

Since '1'='1' is always true, the attacker could gain unauthorized access to the application.

How an SQLi attack happens

1. Vulnerable query

Example of a vulnerable query where user input is directly used in the query.

```python
# Python code with an insecure SQL query
def login(username, password):
    query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}';"
    # Execute the query (insecurely)
    # Assume some function execute_query(query) is called here
    return execute_query(query)
```

Figure 5:Vulnerable SQL injection query

2. SQL injection attack

SQL is injected into the user input field of an *authentication page*

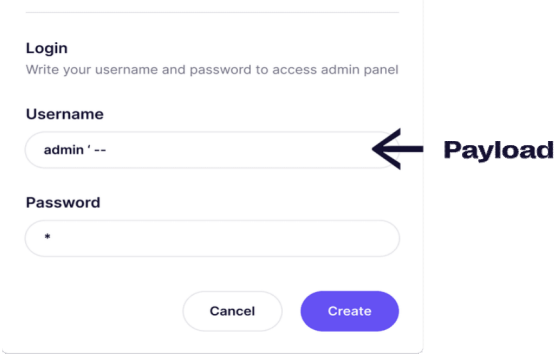

Figure 6: Injection into user input field

3. Query run with payload *Payload comments out the password check so* the *query ignores this step*

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 4 Volume No.9 July - August – 2025
DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

```
# SQL payload injected
def login(username, password):
    query = f"SELECT * FROM users WHERE username = 'admin' -- AND password = '{password}';"
    # Execute the query (insecurely)
    # Assume some function execute_query(query) is called here
    return execute_query(query)
```

Figure 7:SQL injection query

SQL injection by the numbers:
6.7% of all vulnerabilities found in open-source projects are SQLi
10% for closed-source projects!
An increase in the total number of SQL injection in open-source projects (CVE's that involve SQLi) from 2264 *(2023)* to 2400 *(2024)* is expected. (refer Figure 5 and 6)As a percentage of all vulnerabilities, SQL injection is getting less popular: a decrease of 14% and 17% for open-source and closed-source projects respectively from 2023 to 2024 Over 20% of closed source projects scanned are vulnerable to SQL injection when they first start using security toolingFor organizations vulnerable to SQL injection, the average number of SQL injection sites is nearly 30 separate locations in the code. As we can seen in the figure 8 the government database is the most highly targeted and interesting for the attackers which having 22.8% of overall attack , retails 9.1%,finance sector 19.0%, healthcare 137% ,Ecommerce 11.4%, Technology 16.7% and Education 7.2% attack scenario.

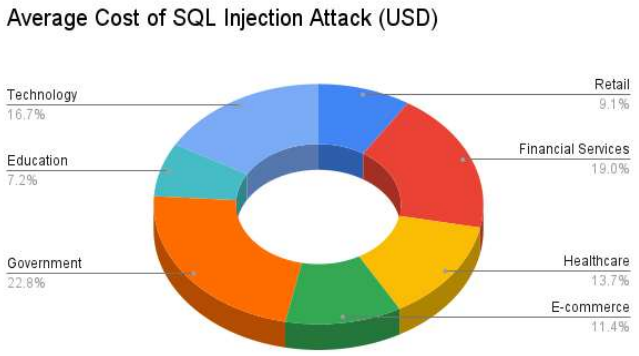

Figure 8: Sectore wise SQL injection attack attack
An analysis conducted by Aikido Security on SQL injection (SQLi) vulnerabilities revealed a continued prevalence of this critical issue across both open-source and closed-source software in 2023 and 2024. In 2024, SQLi accounted for 6.7% of all vulnerabilities identified in open-source packages and 10% in closed-source projects. While these figures represent a 14% decrease in open-source and a 17% decrease in closed-source software compared to 2023, the absolute number of SQLi vulnerabilities remains concerning.
Specifically, the total number of SQLi vulnerabilities in open-source projects is projected to increase from 2,264 in 2023 to over 2,400 by the end of 2024, indicating that the overall volume of vulnerabilities continues to rise despite proportional improvements. These findings underscore the persistence of SQL injection vulnerabilities—despite being one of the most well-understood and preventable classes of security flaws—highlighting the ongoing need for secure coding practices, improved developer education, and automated vulnerability detection and prevention mechanisms.

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 4 Volume No.9 July - August – 2025
DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

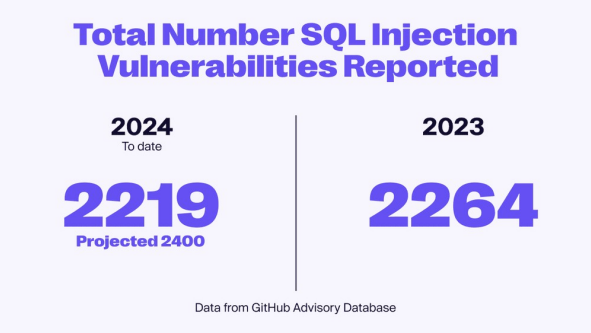

Figure 9:Total number of SQL injection on year 2023-24 report

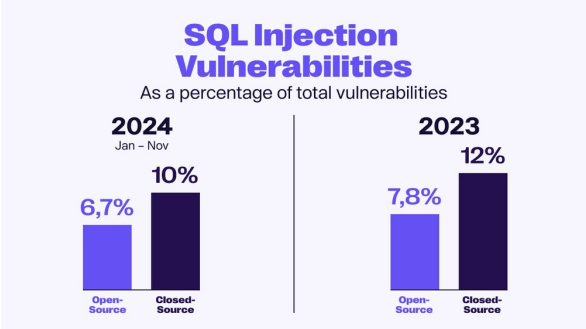

Figure 10: The percentile of open and closed source for SQL injection attack .

Types of SQL Injection

There are several types of SQL Injection attacks, each with different methods of exploiting the vulnerability. These include:

1. *In-band SQL Injection*

In-band SQL Injection is the most common type, where the attacker sends malicious SQL queries directly through the application interface. This method allows attackers to extract sensitive information or manipulate the database.

Example:

An attacker might use the following query in an online login form to extract all users' details:

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

This query bypasses the authentication system and logs in as an admin by making 1=1 always true.

2. *Error-based SQL Injection:*

This type of SQL injection exploits error messages generated by the database. Attackers can use the information provided in error messages to learn about the database structure and craft more sophisticated attacks.

Example:

SELECT * FROM users WHERE id = 1' ;

An error message could reveal details about the database schema, allowing the attacker to refine their attack.

3. *Blind SQL Injection:*

In blind SQL injection, the attacker does not receive error messages but can infer information about the database by observing the behavior of the application. The attacker uses boolean conditions to test various aspects of the database.

Example:

SELECT * FROM users WHERE id = 1 AND 1=1;

If the response is different when 1=1 is changed to 1=0, the attacker can infer that the query was successful, allowing them to gather information about the database.

International Journal of Engineering Technology and Management Sciences
Website: ijetms.in Issue: 4 Volume No.9 July - August – 2025
DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

4. *Out-of-band SQL Injection:*
Out-of-band SQL injection relies on the attacker using a different communication channel to exfiltrate data from the database. This type of attack is less common but can be very effective.
Example:
SELECT * FROM users WHERE id = 1; -- ;
The attacker might direct the database to send a DNS request or HTTP request with the extracted data.
5. *Time-based Blind SQL Injection:*
In this form of blind SQL injection, the attacker sends a query that causes a time delay (e.g., using SLEEP), allowing them to infer whether the query was true or false based on the response time.
Example:
SELECT * FROM users WHERE id = 1 AND SLEEP(5);
If the query takes 5 seconds to execute, the attacker knows that the query is true.

Detecting SQL Injection Vulnerabilities
To detect SQL injection vulnerabilities, consider the following:
Input validation testing: Test inputs by inserting special characters like --, ;, ', or " to see if they cause errors or unintended behavior.
Automated tools: Use tools like SQLMap, Burp Suite, or OWASP ZAP to scan for vulnerabilities.
Review source code: Inspect source code for insecure coding practices such as concatenating user inputs directly into SQL queries.
Monitor error messages: Unexpected or detailed error messages can indicate that the application is vulnerable.
Penetration testing: Regularly perform penetration testing to identify security gaps.

Preventing SQL Injection Attacks
Apparently, there isn't enough information on the internet just yet on how to prevent SQL injection so here is an overview of the options available in 2025:
There are several best practices to prevent SQL injection attacks:
1. *Use Prepared Statements and Parameterized Queries:*
Prepared statements and parameterized queries ensure that user inputs are treated as data rather than part of the SQL query. This approach eliminates the risk of SQL injection.
Example in PHP (using MySQLi):
$stmt = $conn->prepare("SELECT * FROM users WHERE username = ? AND password = ?");
$stmt->bind_param("ss", $username, $password); $stmt->execute();
2. *Employ Stored Procedures:*
Stored procedures are predefined SQL queries stored in the database. These procedures can help prevent SQL injection because they don't dynamically construct SQL queries.
Example:
```
CREATE PROCEDURE GetUserByUsername (IN username VARCHAR(50))
BEGIN
  SELECT * FROM users WHERE username = username;
END;
```
3. *Whitelist Input Validation:*
Ensure that user inputs are validated before being used in SQL queries. Only allow certain characters and patterns, such as alphanumeric input, for fields like usernames or email addresses.
4. *Use ORM Frameworks:*
Object-Relational Mapping (ORM) frameworks like Hibernate or Entity Framework can help prevent SQL injection by automatically handling query generation, preventing dynamic query construction.

DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

5. *Restrict Database Privileges:*
Grant the minimum required database permissions to users. Ensure that applications can only perform necessary actions (e.g., SELECT, INSERT), and restrict permissions like DROP TABLE or ALTER.
6. *Error Handling:*
Configure the database and application to not display detailed error messages to the user. Instead, log errors internally and display generic error messages to end users.
7. *Avoid Dynamic SQL Where Possible:*
Dynamic SQL generation through string concatenation is highly vulnerable to SQL injection. Whenever possible, stick to static, pre-defined queries and stored procedures that don't rely on user-generated content for SQL structure.
8. *Allow listing and escaping:*
In some cases, you cannot avoid Dynamic SQL, such as when querying dynamic tables, or when you want to order by a user-defined column and direction. In those cases you have no other option than to rely on regular expression allowlisting or escaping. Escaping is taking user input that contains dangerous characters used in code like '>' and turning them into a safe form. Ether by adding backslashes before them or transforming them into a symbol code. Note that this is different not only for each database type but can also depend on connection settings such as charset.
9. *Implement an in-app firewall:*
An in-app firewall monitors traffic and activity inside your application and can detect attacks including injection and SQLi. This is more effective than a traditional WAF as it sits inside your application and is able to tokenize expected queries and block requests that change the command structure of the query.

Conclusion:
SQL Injection remains one of the most pervasive and dangerous vulnerabilities affecting modern web applications. Despite decades of awareness and advances in web security practices, SQLi continues to compromise data integrity, confidentiality, and availability across industries. Through real-world case studies—from Heartland Payment Systems in 2008 to recent breaches in 2025—it is evident that both legacy systems and newly developed platforms remain at risk if proper security measures are not implemented.
The persistence of SQLi vulnerabilities, as highlighted by ongoing statistical analysis, underscores a critical gap between known best practices and actual implementation. Factors such as improper input validation, lack of prepared statements, misconfigured systems, and limited use of secure development tools contribute to this continuing risk.
Effective prevention strategies include the consistent use of parameterized queries, server-side input validation, secure coding frameworks, and advanced tooling such as SAST, DAST, and in-app firewalls. Combining these techniques with least-privilege database access, proper error handling, and secure development lifecycle integration can significantly reduce the attack surface. SQL injection evolves in complexity and attackers continue to exploit it as low-hanging fruit, the responsibility lies with developers, security teams, and organizations to embed security deeply within application architectures. Future research and innovation must focus not only on detection and response but also on proactive and automated mitigation to ensure that this avoidable vulnerability is finally brought under control.

Reference :
1. The Bobby Tables site (inspired by the XKCD webcomic) has numerous examples in different languages of parameterized Prepared Statements and Stored Procedures
2. OWASP SQL Injection Prevention Cheat Sheet.
3. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

DOI:10.46647/ijetms.2025.v09i04.003 ISSN: 2581-4621

4. https://www.indusface.com/blog/how-to-stop-sql-injection/ SQL injection attack: Detection, prioritization & prevention
5. Alan Paul, Vishal Sharma, Oluwafemi Olukoya * *School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, United Kingdom.*
6. Fortra Security. SQL Injection Vulnerability in FileCatalyst Workflow 5.1.6 Build 135 (and earlier). 2024, Fortra, https://www.fortra.com/security/advisory/fi-2024-008.
7. Horseman J. CVE-2024-29824 deep dive: Ivanti EPM SQL injection remote code execution vulnerability. 2024, horizon3.ai, https://www.horizon3.ai/attackresearch/ attack-blogs/cve-2024-29824-deep-dive-ivanti-epm-sql-injectionremote-code-execution-vulnerability/.
8. Zhu Z, Jia S, Li J, Qin S, Guo H. SQL injection attack detection framework based on HTTP traffic. In: Proceedings of the ACM turing award celebration conference-China. 2021 .
9. Irungu J, Graham S, Girma A, Kacem T. Artificial intelligence techniques for SQL injection attack detection. In: Proceedings of the 2023 8th international conference on intelligent information technology. 2023.
10. Mitigation from SQL Injection Attacks on Web Server using Open Web Application Security Project Framework Fadlila, I. Riadib, M. A. Mu'min*b
11. Wang B, Yao Y, Shan S, Li H, Viswanath B, Zheng H, et al., editors. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. 2019 IEEE Symposium on Security and Privacy (SP); 2019: IEEE. 10.1109/SP.2019.00031
12. 2. Bora A, Bezboruah T. Investigation on reliability estimation of loosely coupled software as a service execution using clustered and non-clustered web server. International Journal of Engineering, Transactions A: Basics,. 2020;10.5829/ije.2020.33.01a.09
13. 3. Abdullah HS. Evaluation of open source web application vulnerability scanners. Academic Journal of Nawroz University. 2020;9(1):10.25007/ajnu.v9n1a532.
14. **Case-based Explanation of Classification Models for the Detection of SQL Injection Attacks**Juan A. Recio-Garcia*3*, Mauricio G. Orozco-del-Castillo*1,2*,* and Jose A. Soladrero*1,*
15. Detection of SQL Injection Attack Using Machine *2* M. Alghawazi, D. Alghazzawi, S. Alarifi,.
16. Learning Techniques: A Systematic Literature Review, Journal of Cybersecurity and Privacy 2 (2022) 764–777. doi:10.3390/jcp2040039.
17. N. Capuano, G. Fenza, V. Loia, C. Stanzione, Explainable artificial intelligence in cybersecurity: A survey, IEEE Access 10 (2022) 93575–936000