WWW.IJRPR.COM

**International Journal of Research Publication and Reviews**

Journal homepage: www.ijrpr.com ISSN 2582-7421

# SQL Injection and Prevention

***S Yaswanthraj, Abinaya M, Kavya S, Janaki R***

Department of Computer Science and Engineering (Cyber Security)
Bachelor of Engineering, Sri Shakthi Institute of Engineering and Technology
DOI: https://doi.org/10.55248/gengpi.5.0624.1438

**ABSTRACT –**

SQL injection (SQLi) remains one of the most prevalent and damaging attack vectors targeting web applications today. This abstract explores the nature of SQL injection attacks, their potential impacts, and various prevention mechanisms to mitigate this security threat.SQL injection occurs when an attacker manipulates input fields of a web application to execute arbitrary SQL queries against the underlying database. Through these malicious queries, attackers can extract sensitive information, modify or delete data, and even gain unauthorized access to the entire database. The consequences of successful SQL injection attacks can range from data breaches and financial losses to reputational damage for affected organizations.To prevent SQL injection attacks, several defense mechanisms have been developed. These include input validation, parameterized queries, stored procedures, and the use of web application firewalls (wafs). Input validation involves thoroughly validating and sanitizing user input to ensure that only expected data is accepted by the application.

## 1. INTRODUCTION

SQL injection (SQLi) stands as a persistent and prevalent threat to the security of web applications, exploiting vulnerabilities in database systems to compromise sensitive data and undermine the integrity of systems. This introduction provides a foundational understanding of SQL injection, its implications, and the critical importance of implementing effective prevention measures. By comprehensively examining the nature of SQL injection vulnerabilities and their potential impacts, organizations can better appreciate the urgency of addressing this security concern. SQL injection is a cyber attack technique used by hackers to exploit vulnerabilities in web applications that interact with databases. By inserting malicious SQL code into input fields, attackers can manipulate database queries, potentially gaining unauthorized access to sensitive information or executing unauthorized actions on the database server. SQL injection attacks pose a significant threat to the security of web applications and can lead to data breaches, unauthorized access, and system compromise. Understanding and mitigating SQL injection vulnerabilities is essential for ensuring the security of web applications and protecting against potential exploitation.. SQL injection, short for Structured Query Language injection, is a prevalent and potentially devastating cyber attack technique that exploits vulnerabilities in web applications. These vulnerabilities often arise due to improper handling of user-supplied data in SQL queries. Attackers take advantage of this weakness by injecting malicious SQL code into input fields, such as login forms, search boxes, or URL parameters. Once injected, this malicious code can manipulate the SQL queries executed by the web application's backend database. This manipulation can lead to various malicious outcomes, including unauthorized access to sensitive data, data manipulation or deletion, and even complete control over the database server. SQL injection, short for Structured Query Language injection, is a prevalent and potentially devastating cyber attack technique that exploits vulnerabilities in web applications. These vulnerabilities often arise due to improper handling of user-supplied data in SQL queries. Attackers take advantage of this weakness by injecting malicious SQL code into input fields, such as login forms, search boxes, or URL parameters

## II. LITERATURE REVIEW

SQL injection remains a persistent and critical security threat to web applications and databases. Over the years, extensive research has been conducted to understand the various aspects of SQL injection, its detection, prevention, and mitigation strategies. This literature review provides an overview of key findings and contributions in this field.

### *2.1 Historical Context and Evolution:*

Early research on SQL injection dates back to the late 1990s when it was first identified as a serious vulnerability.Initial studies focused on understanding the underlying mechanisms of SQL injection and its impact on web application security.

Subsequent research explored the evolution of SQL injection techniques and the emergence of more sophisticated attack vectors.

### *2.2 Detection Techniques:*

Researchers have proposed various techniques for detecting SQL injection vulnerabilities in web applications. Static analysis approaches involve analyzing source code or bytecode to identify potential injection points and vulnerable code patterns. Dynamic analysis techniques, such as black-box and white-box testing, involve sending crafted input to the application and observing its behavior to detect SQL injection vulnerabilities. Hybrid approaches combine static and dynamic analysis methods to achieve more comprehensive vulnerability detection.

### *2.3 Prevention and Mitigation Strategies:*

Numerous strategies have been proposed to prevent and mitigate SQL injection attacks.Input validation and sanitization techniques aim to filter user-supplied input to prevent malicious SQL code injection.

## III. SQL INJECTION TECHNIQUES

### *3.1 Union-Based SQL Injection:*

This technique involves injecting a UNION statement into an SQL query to combine the result of the original query with the injected query. By manipulating the number of columns selected in the injected query, an attacker can retrieve data from other database tables

### *3.2 Boolean-Based SQL Injection:*

In this technique, the attacker exploits the application's response to Boolean-based SQL queries. By injecting SQL code that evaluates to true or false, the attacker can infer information about the database or manipulate the application's behavior.

### *3.3 Error-Based SQL Injection:*

Error-based SQL injection exploits error messages generated by the database server to extract information about the structure and content of the database. Attackers inject malicious SQL code that intentionally triggers errors, revealing details such as table names, column names, and sensitive data.

### *3.4 Time-Based SQL Injection:*

Time-based SQL injection involves injecting SQL code that introduces delays in the execution of the query. By observing the application's response time, an attacker can infer whether the injected condition is true or false, allowing them to extract information from the database.

### *3..5 Out-of-Band (OOB) SQL Injection:*

This technique involves exploiting vulnerabilities to establish a communication channel between the application server and an external server controlled by the attacker. The attacker can then exfiltrate data or execute commands through this channel.

### *3.6 Second-Order SQL Injection:*

Second-order SQL injection occurs when user input is stored in the database and later used in an SQL query without proper validation or sanitization. Attackers can inject malicious code that remains dormant until triggered by specific conditions, making detection more challenging.

### *3.7 Blind SQL Injection:*

Blind SQL injection techniques exploit vulnerabilities where the application does not directly reveal the results of SQL queries. Attackers use techniques such as boolean-based or time-based inference to extract information indirectly, without seeing the actual query results.

### *3.8 Stored Procedure Injection:*

In systems that use stored procedures, attackers can inject malicious code into parameterized stored procedure calls. If the stored procedure executes dynamic SQL without proper validation, it can lead to SQL injection vulnerabilities.

### *3.9 Function Call Injection:*

Similar to stored procedure injection, function call injection involves injecting malicious code into function calls with in SQL queries. If the function executes dynamic SQL or performs database operations, it can be vulnerable to SQL injection.

***3.10 Metadata Injection:***

Attackers can inject SQL code that retrieves metadata about the database, such as table names, column names, and data types. This information can be used to further refine and execute subsequent attacks.

## IV. PREVENTING METHODOLOGIES

Preventing vulnerabilities and attacks requires a proactive approach that involves implementing security measures and best practices throughout the software development lifecycle. Here are some methodologies and practices for preventing vulnerabilities:

***4.1Secure Development Lifecycle (SDL):***

Incorporate security into every phase of the software development process, including requirements gathering, design, implementation, testing, deployment, and maintenance.

***4.2 Security Training and Awareness:***

Provide regular security training and awareness programs for developers, testers, and other stakeholders to educate them about common vulnerabilities and best practices.

***4.3 Code Reviews and Static Analysis:***

Perform regular code reviews to identify security flaws and vulnerabilities. Utilize automated static analysis tools to scan code for known vulnerabilities and adherence to secure coding practices.

***4.4 Dynamic Application Security Testing (DAST):***

Use dynamic analysis tools to assess the security of running applications by simulating attacks and identifying vulnerabilities in real-time.

***4.7 Penetration Testing:***

Conduct regular penetration testing exercises to simulate real-world attacks and identify vulnerabilities that may not be detected by automated tools. This helps in assessing the effectiveness of security controls and defenses.

***4.8 Vulnerability Management:***

Implement a robust vulnerability management program to regularly scan systems and applications for known vulnerabilities, prioritize them based on risk, and apply patches or mitigations in a timely manner.

***4.9 Secure Coding Guidelines:***

Establish and enforce secure coding standards and guidelines based on industry best practices and security standards (e.g., OWASP Top 10, CWE/SANS Top 25) to help developers write secure code.

***4.10 Least Privilege Principle:***

Follow the principle of least privilege by granting users and processes only the minimum level of access or permissions required to perform their tasks.

***4.11 Secure Configuration Management:***

Ensure that servers, databases, and other components are securely configured and hardened according to industry best practices and vendor recommendations.

## V. TESTING AND REMIDIATION

Testing and remediation are critical aspects of addressing SQL injection vulnerabilities and ensuring the security of web applications. Let's delve deeper into these topics:

***5.1. Penetration Testing:***

Conduct penetration tests specifically targeting SQL injection vulnerabilities. This involves attempting to exploit potential vulnerabilities to gain unauthorized access to the database or manipulate its data.

***5.2 Static Code Analysis:***

Utilize static code analysis tools to scan the source code for potential SQL injection vulnerabilities. These tools can identify insecure coding practices and provide recommendations for remediation.

***5.3 Dynamic Application Security Testing (DAST):***

Use DAST tools to dynamically assess the security of web applications in real-time. These tools can detect SQL injection vulnerabilities by sending malicious inputs and analyzing the application's responses.

***5.4 Manual Code Review:***

Perform manual code reviews by experienced developers or security experts to identify SQL injection vulnerabilities that automated tools may overlook. This involves examining the application's source code line by line to identify insecure coding patterns.

***5.5 Fuzz Testing:***

Employ fuzz testing techniques to systematically test input fields and parameters for unexpected or malicious inputs that could trigger SQL injection vulnerabilities.

***5.6 Parameterized Queries:***

Rewrite SQL queries to use parameterized queries (prepared statements) instead of concatenating user input directly into SQL statements. This helps prevent SQL injection by separating data from the query logic.

***5.7 Input Validation and Sanitization:***

Implement robust input validation and sanitization routines to ensure that user input conforms to expected formats and doesn't contain malicious characters or SQL code.

***5.8 Escaping User Input:***

If parameterized queries are not feasible, ensure that user input is properly escaped before being included in SQL statements.

***5.9 ORMs (Object-Relational Mapping):***

Use ORM libraries or frameworks that automatically handle SQL query generation and parameterization, reducing the risk of SQL injection vulnerabilities.

***5.10 Database Firewall:***

Deploy a database firewall to monitor and filter SQL queries before they reach the database. This can help detect and block SQL injection attacks in real-time.

***5.11 Security Training:***

Provide security training for developers to raise awareness of SQL injection vulnerabilities and teach best practices for writing secure code. This includes emphasizing the importance of input validation, parameterized queries, and secure coding practices.

***5.12 Patch Management:***

Keep web application frameworks, libraries, and database management systems up-to-date with the latest security patches to address known vulnerabilities that could be exploited by attackers.

***5.13 Regular Audits:***

Conduct regular security audits and assessments to identify and remediate any newly discovered SQL injection vulnerabilities. This includes retesting the application after implementing remediation measures to ensure that the vulnerabilities have been effectively mitigated.

## VI. IMPACT OF SQL INJECTION

***6.1 Data Leakage:***

Attackers can extract sensitive information from the database, such as usernames, passwords, credit card numbers, and personal identifiable information (PII).

***6.2 Data Manipulation:***

Malicious SQL injection queries can modify or delete data within the database, leading to data loss, corruption, or unauthorized changes to critical information.

***6.3 Unauthorized Access:***

SQL injection attacks can bypass authentication mechanisms, granting unauthorized access to restricted parts of the application or administrative privileges within the database.

***6.4 Denial of Service (DoS):***

In some cases, SQL injection attacks can be used to execute resource-intensive queries, leading to a slowdown or unavailability of the database server, resulting in a denial of service for legitimate users.

***6.5 Reputation Damage:***

A successful SQL injection attack can damage an organization's reputation, leading to loss of customer trust, legal consequences, and financial repercussions.

***6.6 Regulatory Compliance Violations:***

Depending on the type of data involved, a SQL injection attack may result in violations of regulatory requirements such as GDPR, HIPAA, or PCI DSS, leading to fines and legal penalties.

## VII. PROPOSED SYSTEM

***7.1 Frontend:***

The frontend of the web application is responsible for presenting the user interface to the clients.It should implement client-side input validation to ensure that only valid data is sent to the backend.Utilize modern frontend frameworks such as React.js, Angular, or Vue.js to build a responsive and interactive user interface.

***7.2 Backend:***

The backend is responsible for handling client requests, processing business logic, and interacting with the database.Implement an API layer using a server-side framework like Node.js with Express, Django, Flask, ASP.NET Core, or Spring Boot.Use ORM (Object-Relational Mapping) libraries such as Sequelize (Node.js), Django ORM (Python), SQLAlchemy (Python), Entity Framework (C#), or Hibernate (Java) to interact with the database securely.Implement input validation and sanitization on the server-side to prevent SQL injection attacks.

***7.3 Database:***

Choose a secure and well-established relational database management system (RDBMS) such as MySQL, PostgreSQL, Microsoft SQL Server, or Oracle.Configure the database server securely, including proper authentication, access controls, and encryption of sensitive data.Utilize database features such as parameterized queries, stored procedures, and prepared statements to prevent SQL injection vulnerabilities.Regularly apply security patches and updates to the database software to address known vulnerabilities.

### *7.4 Security Measures:*

Implement a Web Application Firewall (WAF) to monitor and filter incoming HTTP requests for suspicious patterns indicative of SQL injection attacks.Enable logging and monitoring of database activities to dtect and respond to SQL injection attempts and other security incidents.Conduct regular security audits, code reviews, and penetration testing to identify and address potential vulnerabilities in the application code and infrastructure.Provide training and education to developers and administrators on secure coding practices, including the prevention of SQL injection and other common security threats.

### *7.5 Infrastructure:*

Deploy the web application on a secure and scalable cloud platform such as Amazon Web Services (AWS), Microsoft Azure, or Google Cloud Platform (GCP).Utilize containerization technologies such as Docker and orchestration tools like Kubernetes for managing application deployment and scalability.Implement secure network configurations, including firewalls, VPNs, and encryption protocols, to protect data in transit and at rest.

## VIII. LOGGING AND MONITORING

### *8.1 Types of Logs:*

Understand the different types of logs generated by systems, applications, and network devices, including:

System Logs: Records system-level events such as startup/shutdown, authentication events, and hardware errors.Application Logs: Capture application-specific events, errors, and user activities.Security Logs: Document security-related events such as login attempts, access control changes, and detected threats.

Network Logs: Track network traffic, including connection attempts, data transfers, and intrusion attempts.

### *8.2 Logging Best Practices:*

Log Everything of Value: Ensure that critical events, errors, and security-related activities are logged to provide comprehensive visibility into system and application behavior.

Standardized Log Formats: Adopt standardized log formats and conventions to facilitate log analysis and correlation across different systems and applications.

Log Integrity and Confidentiality: Protect log data from tampering and unauthorized access by implementing measures such as encryption, access controls, and integrity checks.

Retention Policies: Define retention policies for log data to balance storage requirements with regulatory compliance and incident investigation needs.

### *8.3 Monitoring Tools and Technologies:*

SIEM (Security Information and Event Management): SIEM platforms aggregate, correlate, and analyze log data from various sources to detect security incidents and support incident response activities.

IDS/IPS (Intrusion Detection/Prevention Systems): IDS/IPS solutions monitor network traffic for suspicious patterns or signatures indicative of malicious activity and can take automated actions to block or mitigate threats.

Endpoint Detection and Response (EDR): EDR solutions provide real-time monitoring and response capabilities on individual endpoints, enabling the detection and containment of threats at the endpoint level.

User and Entity Behavior Analytics (UEBA): UEBA solutions analyze user and entity behavior patterns to identify anomalous activities indicative of insider threats or compromised accounts.

### *8.4 Alerting and Response:*

Threshold-based Alerts: Configure threshold-based alerts for critical events and anomalies to notify security personnel promptly.

Automated Response: Implement automated response mechanisms to mitigate identified threats or vulnerabilities in real-time, reducing the manual intervention required for incident response.

Incident Response Playbooks: Develop incident response playbooks outlining predefined procedures and response actions for different types of security incidents, including those detected through logging and monitoring.

Regular Log Analysis: Conduct regular log analysis and review to identify patterns, trends, and emerging threats, informing proactive security measures and controls.

Incident Post-Mortems: Conduct post-incident reviews and analysis to identify gaps in logging and monitoring coverage, refine detection rules, and improve incident response procedures.

## IX. Security Headers

### *9.1Content Security Policy (CSP):*

Purpose: CSP helps prevent cross-site scripting (XSS) attacks by defining a whitelist of trusted sources for content such as scripts, stylesheets, and media files.

Example Header: Content-Security-Policy: script-src 'self' https://example.com;

### *9.2 X-Content-Type-Options:*

Purpose: Prevents MIME type sniffing, which can lead to content-type confusion attacks by ensuring that browsers adhere to the declared content type.

Example Header: X-Content-Type-Options: nosniff

### *9.3 X-XSS-Protection:*

Purpose: Enables or configures the built-in XSS filter in modern web browsers to block or sanitize potentially malicious scripts.

Example Header: X-XSS-Protection: 1; mode=block

### *9.4 Strict-Transport-Security (HSTS):*

Purpose: Enforces the use of HTTPS by instructing browsers to only access the website over a secure connection (HTTPS) and automatically redirect HTTP requests to HTTPS.

Example Header: Strict-Transport-Security: max-age=31536000; includeSubDomains; preload

### *9.5 Referrer-Policy:*

Purpose: Controls how much information is included in the Referer header when navigating from one site to another, helping to prevent information leakage.

Example Header: Referrer-Policy: no-referrer

### *9.6 Feature-Policy:*

Purpose: Allows web developers to control which browser features and APIs can be used by a web application, reducing the attack surface for potential exploits.

Example Header: Feature-Policy: camera 'none'; microphone 'none'

### *9.7 Frame Options:*

Purpose: Prevents clickjacking attacks by controlling whether a web page can be loaded inside a frame or frame. Example Header: X-Frame-Options: DENY

### *9.8 Cross-Origin Resource Sharing (CORS):*

Purpose: Controls access to resources from different origins to mitigate cross-origin HTTP request vulnerabilities.Example Header: Access-Control-Allow-Origin: https://example.com

### *9.9 Cache-Control and Pragma:*

Purpose: Controls caching behavior to prevent sensitive information from being cached by proxies or the client's browser.Example Header: Cache-Control: no-cache, no-store, must-revalidate

## X. Third-Party Library and Framework Security

***10.1 Risk Assessment:***

Vulnerability Management: Regularly monitor vulnerability databases and security advisories to identify known vulnerabilities in third-party libraries and frameworks.

Risk Prioritization: Assess the severity and potential impact of identified vulnerabilities to prioritize patching and mitigation efforts accordingly.

***10.2 Dependency Management:***

Version Control: Maintain an up-to-date inventory of third-party dependencies used in the project, including their versions and associated vulnerabilities.

Dependency Scanning: Utilize automated dependency scanning tools to identify outdated or vulnerable dependencies and enforce version constraints.

***10.3 Security Review:***

Code Review: Perform security-focused code reviews of third-party library code to identify potential security flaws or weaknesses.

Static Analysis: Use static code analysis tools to identify security vulnerabilities and coding errors in third-party library code.

***10.4 Secure Configuration:***

Secure Defaults: Configure third-party libraries and frameworks with secure defaults to minimize the risk of misconfigurations leading to security vulnerabilities.

Least Privilege: Limit the privileges and permissions granted to third-party libraries and frameworks to reduce the potential impact of security compromises.

***10.5 Secure Integration:***

Secure Communication: Ensure secure communication channels when interacting with third-party services or APIs to prevent eavesdropping, man-in-the-middle attacks, and data breaches.

Input Validation: Validate and sanitize inputs from third-party libraries and frameworks to mitigate injection attacks and other security vulnerabilities.

***10.4 Patch Management:***

Timely Updates: Stay vigilant for security updates and patches released by third-party library maintainers and vendors and apply them promptly to mitigate known vulnerabilities.

Automated Patching: Implement automated patch management processes to streamline the deployment of security updates for third-party dependencies.

***10.5 Vendor Risk Management:***

Vendor Assessment: Assess the security practices and track record of third-party library vendors and maintainers to evaluate the overall risk associated with using their software.

Contractual Obligations: Include security requirements and expectations in contracts and agreements with third-party vendors to ensure accountability and compliance with security standards.

***10.6 Continuous Monitoring:***

Runtime Monitoring: Implement runtime monitoring and anomaly detection mechanisms to detect and respond to security incidents and abnormal behavior resulting from vulnerabilities in third-party libraries and frameworks.

Security Testing: Conduct regular security testing, including penetration testing and vulnerability scanning, to identify and address security weaknesses introduced by third-party dependencies.

## XI. CONCLUTION

SQL injection is a prevalent and potentially devastating attack vector in web applications, where attackers inject malicious SQL queries to manipulate the database and access unauthorized data or perform unauthorized actions. To effectively prevent SQL injection attacks, developers and organizations must implement robust security measures throughout the development lifecycle.

Parameterized Queries: Always use parameterized queries (prepared statements) with bound parameters instead of dynamically constructing SQL queries with user input.

Input Validation and Sanitization: Validate and sanitize all user inputs to ensure they conform to expected formats and do not contain malicious code. Use whitelisting rather than blacklisting to define acceptable input patterns.

Least Privilege Principle: Implement the principle of least privilege, granting the minimum necessary permissions to database users and restricting access to sensitive operations and data.

Escaping: If dynamic SQL construction is unavoidable, properly escape user input using appropriate escaping mechanisms provided by the programming language or framework being used.

Web Application Firewalls (WAFs): Deploy WAFs to filter and block malicious SQL injection attempts at the network perimeter.

Education and Training: Educate developers about secure coding practices and common attack vectors, including SQL injection, through training programs and resources.

In conclusion, while SQL injection remains a significant threat to web application security, its risks can be mitigated through proactive measures such as using parameterized queries, input validation, least privilege principles, and ongoing security audits. By incorporating these practices into the development process, organizations can significantly reduce the likelihood of SQL injection attacks and enhance the overall security posture of their web applications.

## XII. REFERENCES

[1] Hlaing, Zar Chi Su Su, and Myo Khaing. "A detection and prevention technique on sql injection attacks." 2020 IEEE Conference on Computer Applications (ICCA). IEEE, 2020.

[2] Hlaing, Z. C. S. S., & Khaing, M. (2020, February). A detection and prevention technique on sql injection attacks. In 2020 IEEE Conference on Computer Applications (ICCA) (pp. 1-6). IEEE

[3] Hlaing, Zar Chi Su Su, and Myo Khaing. "A detection and prevention technique on sql injection attacks." In 2020 IEEE Conference on Computer Applications (ICCA), pp. 1-6. IEEE, 2020.

[4] Hlaing, Z.C.S.S. and Khaing, M., 2020, February. A detection and prevention technique on sql injection attacks. In 2020 IEEE Conference on Computer Applications (ICCA) (pp. 1-6). IEEE.

[5] Hlaing ZC, Khaing M. A detection and prevention technique on sql injection attacks. Nasereddin, M., ALKhamaiseh, A., Qasaimeh, M., & Al-Qassas, R. (2023). A systematic review of detection and prevention techniques of SQL injection attacks. Information Security Journal: A Global Perspective, 32(4), 252-265.1-6). IEEE.

[6] Kareem, Fairoz Q., et al. "SQL injection attacks prevention system technology." Asian Journal of Research in Computer Science 6.15 (2021): 13-32.

APA

[7] Kareem, F. Q., Ameen, S. Y., Salih, A. A., Ahmed, D. M., Kak, S. F., Yasin, H. M., ... & Omar, N. (2021). SQL injection attacks prevention system technology. Asian Journal of Research in Computer Science, 6(15), 13-32.

Chicago

[8] Kareem, Fairoz Q., Siddeeq Y. Ameen, Azar Abid Salih, Dindar Mikaeel Ahmed, Shakir Fattah Kak, Hajar Maseeh Yasin, Ibrahim Mahmood Ibrahim, Awder Mohammed Ahmed, Zryan Najat Rashid, and Naaman Omar. "SQL injection attacks prevention system technology." Asian Journal of Research in Computer Science 6, no. 15 (2021): 13-32.

Harvard

[9] Kareem, F.Q., Ameen, S.Y., Salih, A.A., Ahmed, D.M., Kak, S.F., Yasin, H.M., Ibrahim, I.M., Ahmed, A.M., Rashid, Z.N. and Omar, N., 2021. SQL injection attacks prevention system technology. Asian Journal of Research in Computer Science, 6(15), pp.13-32.

Vancouver

[10] Kareem FQ, Ameen SY, Salih AA, Ahmed DM, Kak SF, Yasin HM, Ibrahim IM, Ahmed AM, Rashid ZN, Omar N. SQL injection attacks prevention system technology. Asian Journal of Research in Computer Science. 2021 Jul;6(15):13-32.

[11] Nasereddin, Mohammed, et al. "A systematic review of detection and prevention techniques of SQL injection attacks." Information Security Journal: A Global Perspective 32.4 (2023): 252-265

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

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

[14] Nasereddin M, ALKhamaiseh A, Qasaimeh M, Al-Qassas R. A systematic review of detection and prevention techniques of SQL injection attacks. Information Security Journal: A Global Perspective. 2023 Jul 4;32(4):252-65.

[15] Nasereddin M, ALKhamaiseh A, Qasaimeh M, Al-Qassas R. A systematic review of detection and prevention techniques of SQL injection attacks. Information Security Journal: A Global Perspective. 2023 Jul 4;32(4):252-65.

[16] Rai, A., Miraz, M. M. I., Das, D., & Kaur, H. (2021, April). Sql injection: Classification and prevention. In 2021 2nd International conference on Intelligent Engineering and Management (ICIEM) (pp. 367-372). IEEE.

[17] Rai, Aditya, MD Mazharul Islam Miraz, Deshbandhu Das, and Harpreet Kaur. "Sql injection: Classification and prevention." In 2021 2nd International conference on Intelligent Engineering and Management (ICIEM), pp. 367-372. IEEE, 2021.

[18] Rai, A., Miraz, M.M.I., Das, D. and Kaur, H., 2021, April. Sql injection: Classification and prevention. In 2021 2nd International conference on Intelligent Engineering and Management (ICIEM) (pp. 367-372). IEEE.

[19] Rai A, Miraz MM, Das D, Kaur H. Sql injection: Classification and prevention. In2021 2nd International conference on Intelligent Engineering and Management (ICIEM) 2021 Apr 28 (pp. 367-372). IEEE.

[20] Dasmohapatra, Sabyasachi, and Sushree Bibhuprada B. Priyadarshini. "A Comprehensive Study on SQL Injection Attacks, Their Mode, Detection and Prevention." Proceedings of Second Doctoral Symposium on Computational Intelligence: DoSCI 2021. Springer Singapore, 2022.