JOWIM
International Journal of Wireless and Multimedia Communications
Vol. 3 No. 2 (2026), June 2026, pp. 14-24
ISSN: 3047-2172

Analysis Of Website Security Using Sql Injection Penetration Testing On Damn Vulnerable Web Application

Rostam Ahmad Efendi[a], Ismael[b]

[ab] *Politeknik Negeri Padang, Padang, 25171, Indonesia*

[a*] *rostamahmadefendi@pnp.ac.id*

Abstract— Website security has become one of the primary concerns in the development of information systems due to the increasing number of cyberattacks targeting web-based applications. Among various cyber threats, SQL Injection remains one of the most critical vulnerabilities because it allows attackers to manipulate Structured Query Language (SQL) statements through unsensitized user inputs, potentially resulting in unauthorized access, data leakage, data manipulation, or complete system compromise. This research aims to analyse website vulnerabilities against SQL Injection attacks using a penetration testing approach on the Damn Vulnerable Web Application (DVWA), a deliberately vulnerable web application widely utilized for cybersecurity education and security assessment. The research employed an experimental methodology consisting of reconnaissance, vulnerability identification, exploitation, and vulnerability analysis. Penetration testing was conducted using Burp Suite, browser developer tools, and manually crafted SQL Injection payloads. The testing results demonstrated that the low-security configuration of DVWA was highly susceptible to authentication bypass attacks, allowing unauthorized access to sensitive information. The analysis further revealed that the primary causes of these vulnerabilities were inadequate input validation, dynamic SQL query construction, and the absence of parameterized queries or prepared statements. This study emphasizes the importance of implementing secure coding practices, input validation, parameterized SQL queries, and Web Application Firewall (WAF) technologies to mitigate SQL Injection risks. The findings contribute to cybersecurity education by providing practical evidence of common SQL Injection vulnerabilities and effective mitigation strategies that can be adopted by software developers and information system administrators.

Keywords— *Website Security; SQL Injection; Penetration Testing; DVWA; Cybersecurity.*

Manuscript received 1 Jun. 2026 ; revised 14 Jun 2026; accepted 20 Jun. 2026. Date of publication 30 Jun. 2026

I. INTRODUCTION

The rapid advancement of information technology has significantly transformed the way organizations deliver services, manage information, and conduct business operations. Web-based applications have become fundamental components in various sectors, including government, education, healthcare, finance, and electronic commerce, due to their accessibility, scalability, and cost-effectiveness. However, the increasing dependence on web applications has also expanded the attack surface for cybercriminals, making web application security one of the most critical challenges in modern information systems.

Among numerous web security threats, SQL Injection (SQLi) remains one of the most dangerous and persistent vulnerabilities. SQL Injection occurs when user-supplied input is improperly validated and directly incorporated into SQL statements, allowing attackers to manipulate database queries. Successful exploitation may result in authentication bypass, unauthorized disclosure of confidential information, modification or deletion of database records, privilege escalation, and even complete compromise of information systems. According to the OWASP Top 10 (2021), Injection vulnerabilities continue to rank among the most critical security risks affecting web applications worldwide. Similarly, the MITRE Common Weakness Enumeration classifies SQL Injection as CWE-89, highlighting its severe impact on software security and the necessity of implementing secure coding practices throughout the software development lifecycle.

Modern software development frameworks provide various mechanisms for preventing SQL Injection attacks, including parameterized queries, prepared statements, input validation, and object-relational mapping frameworks. Nevertheless, many legacy applications, educational platforms, and improperly developed web systems continue to expose exploitable SQL Injection vulnerabilities due to insecure programming practices. Consequently, periodic security assessment through penetration

testing has become an essential activity for evaluating the resilience of web applications before they are deployed or exposed to production environments.

Penetration testing is a controlled security evaluation process that simulates real-world cyberattacks to identify exploitable vulnerabilities within information systems. Unlike automated vulnerability scanning, penetration testing attempts to actively exploit identified weaknesses to evaluate their practical impact on confidentiality, integrity, and availability. This approach enables developers and security professionals to understand actual attack scenarios while prioritizing remediation efforts based on measurable security risks.

One of the most widely adopted educational platforms for learning web application security is the Damn Vulnerable Web Application (DVWA). DVWA is intentionally designed with multiple security vulnerabilities, allowing researchers, students, and cybersecurity practitioners to perform penetration testing in a safe and isolated environment. The platform provides several configurable security levels ranging from Low to Impossible, enabling researchers to evaluate how secure coding mechanisms influence application resilience against different attack techniques. Because of its controlled environment and reproducible vulnerabilities, DVWA continues to serve as an effective platform for cybersecurity education, penetration testing training, and secure software development research.

Several previous studies have investigated SQL Injection from different perspectives, including vulnerability detection, secure coding techniques, automated vulnerability scanners, and machine-learning-based detection approaches. Although these studies have significantly contributed to the development of web application security, most primarily emphasize theoretical analysis or automated detection techniques. Comparatively fewer studies provide a comprehensive experimental investigation that combines manual penetration testing, authentication bypass verification, database information disclosure analysis, and security risk evaluation using internationally recognized standards such as OWASP Top 10 (2021), CWE-89, and the Common Vulnerability Scoring System (CVSS v3.1). Furthermore, limited research discusses how practical penetration testing results can be integrated into secure software development practices and DevSecOps implementation.

To address these research gaps, this study proposes a practical penetration testing framework for evaluating SQL Injection vulnerabilities using the Damn Vulnerable Web Application (DVWA). The proposed approach integrates manual SQL Injection exploitation, vulnerability verification, authentication bypass analysis, database information disclosure assessment, and risk evaluation based on OWASP Top 10 (2021), CWE-89, and CVSS v3.1. In addition, this research presents comprehensive mitigation strategies that combine secure coding principles, penetration testing practices, and DevSecOps concepts to strengthen the security posture of web applications.

Specifically, this research aims to:

1. Identify SQL Injection vulnerabilities within the Damn Vulnerable Web Application (DVWA).
2. Evaluate authentication bypass and database information disclosure through manual penetration testing.
3. Assess the severity of identified vulnerabilities using OWASP Top 10 (2021), CWE-89, and CVSS v3.1.
4. Propose effective mitigation strategies based on secure coding principles and secure software development practices.

Accordingly, this study addresses the following research questions:

- RQ1: Which input parameters within DVWA are vulnerable to SQL Injection attacks?
- RQ2: How effective are manual SQL Injection payloads in bypassing authentication mechanisms?
- RQ3: What security impacts result from successful SQL Injection exploitation?
- RQ4: Which mitigation strategies are most effective in reducing SQL Injection risks in web applications?

The findings of this research are expected to contribute to cybersecurity education by providing practical evidence of SQL Injection exploitation and mitigation while supporting software developers, information system administrators, and cybersecurity practitioners in implementing secure coding standards and integrating penetration testing into the Secure Software Development Life Cycle (SSDLC) and DevSecOps practices.

The remainder of this paper is organized as follows. Section II presents the related work and summarizes previous studies on SQL Injection vulnerabilities and penetration testing. Section III describes the research materials, experimental environment, and methodology employed in this study. Section IV presents the experimental results and discussion, including vulnerability analysis, risk assessment, and mitigation strategies. Finally, Section V concludes the paper, discusses its limitations, and outlines directions for future research.

II. RELATED WORK

SQL Injection (SQLi) has remained one of the most critical security threats affecting web applications for more than two decades. Despite significant advancements in secure software development methodologies and defensive technologies, SQL Injection continues to be responsible for numerous data breaches and unauthorized database access incidents worldwide. Consequently, extensive research has been conducted to improve vulnerability detection, penetration testing methodologies, secure coding practices, and automated defense mechanisms.

One of the earliest comprehensive studies on SQL Injection was conducted by Halfond, Viegas, and Orso, who classified SQL

Injection attacks into several categories and introduced corresponding countermeasures for preventing database exploitation. Their work established the foundation for modern SQL Injection research by emphasizing the importance of input validation, query sanitization, and secure programming techniques.

The Open Worldwide Application Security Project (OWASP) continues to identify Injection vulnerabilities as one of the most critical risks in web application security. According to the OWASP Top 10 (2021), Injection attacks remain capable of compromising the confidentiality, integrity, and availability of information systems when user inputs are improperly processed. OWASP strongly recommends implementing parameterized queries, prepared statements, secure input validation, and least-privilege database access as primary mitigation strategies.

Recent studies have focused on improving SQL Injection detection using automated vulnerability scanners, static code analysis, and machine learning techniques. Jain and Gupta presented a comprehensive survey of SQL Injection detection and prevention mechanisms, concluding that combining multiple detection approaches generally improves vulnerability identification accuracy. Similarly, Almseidin et al. evaluated several SQL Injection detection techniques and demonstrated that hybrid approaches provide higher detection effectiveness than single-method solutions.

Machine learning has also been introduced to enhance web application security. Li et al. proposed an intelligent SQL Injection detection framework capable of recognizing malicious SQL patterns through supervised learning algorithms. Their study demonstrated that machine learning techniques can improve detection accuracy for complex attack patterns; however, the approach requires extensive training datasets and computational resources, making practical implementation more challenging in smaller organizations.

In addition to vulnerability detection, several researchers have investigated secure coding practices as an effective defense against SQL Injection attacks. Alwan and Younis demonstrated that prepared statements and parameterized SQL queries remain the most reliable techniques for preventing SQL Injection because they separate executable SQL commands from user-supplied input. Likewise, Howard and LeBlanc emphasized that secure software development should integrate defensive programming principles from the earliest stages of system design rather than relying solely on post-deployment penetration testing.

Although previous studies have significantly advanced SQL Injection prevention and detection techniques, several research gaps remain. First, many existing studies primarily focus on theoretical discussions, automated detection algorithms, or static code analysis without demonstrating practical exploitation in a controlled environment. Second, comparatively few studies integrate manual penetration testing, authentication bypass verification, database information disclosure analysis, and standardized security risk assessment within a single experimental framework. Third, limited research explicitly discusses how penetration testing results can support secure software development practices and DevSecOps implementation.

Unlike previous research, this study emphasizes practical penetration testing using the Damn Vulnerable Web Application (DVWA), an intentionally vulnerable platform widely adopted for cybersecurity education. The proposed research integrates manual SQL Injection exploitation, authentication bypass verification, database information disclosure analysis, qualitative vulnerability assessment, and risk evaluation using OWASP Top 10 (2021), CWE-89, and CVSS v3.1. Furthermore, this research proposes comprehensive mitigation strategies based on secure coding principles, secure software development lifecycle (SSDLC), and DevSecOps practices. Therefore, the study contributes not only to vulnerability identification but also to bridging the gap between penetration testing activities and secure software engineering practices.

Table I summarizes representative studies related to SQL Injection security and highlights the research gap addressed by this study.

Table 1 Comparison Of Previous Studies

Research	Research Focus	Method	Main Findings	Research Gap
Halfond et al. (2006)	SQL Injection Classification	Literature Review	Classification of SQL Injection attacks and countermeasures	No experimental penetration testing
OWASP (2021)	Web Application Security	Security Guideline	Injection remains a critical vulnerability	No practical validation
Jain & Gupta (2021)	SQL Injection Detection	Survey	Detection and prevention techniques	No exploitation analysis
Almseidin et al. (2022)	Detection Evaluation	Experimental Comparison	Hybrid detection techniques improve effectiveness	Limited secure coding discussion
Li et al. (2023)	Machine Learning Detection	Machine Learning	High detection accuracy	Requires large datasets and computational resources
This Research	Practical SQL Injection Assessment	Manual Penetration Testing	Authentication bypass, database information disclosure, OWASP-based risk assessment, and mitigation recommendations	Integrates exploitation, risk assessment, and secure software development practices

Based on the comparison presented above, this study complements previous research by integrating practical penetration testing with standardized vulnerability assessment

and secure software engineering recommendations. The findings are expected to provide practical references for cybersecurity practitioners, educators, researchers, and software developers seeking to improve web application security against SQL Injection attacks.

III. MATERIALS AND METHODS

A. Research Design

This study employed an experimental research design using a manual penetration testing approach to evaluate SQL Injection vulnerabilities in a controlled web application environment. The research was conducted within an isolated laboratory environment to ensure that all penetration testing activities could be performed safely without affecting production systems or violating ethical and legal considerations.

The research methodology follows internationally recognized penetration testing practices, including the Penetration Testing Execution Standard (PTES) and the OWASP Web Security Testing Guide (WSTG). These frameworks provide systematic procedures for identifying, exploiting, verifying, and assessing security vulnerabilities within web applications.

The overall research process consists of six major phases:

1. Experimental Environment Preparation
2. Information Gathering
3. Vulnerability Identification
4. SQL Injection Exploitation
5. Risk Assessment
6. Security Recommendation

Figure 1 illustrates the overall research methodology adopted in this study.

B. Research Architecture

The experimental architecture consists of a client workstation, penetration testing tools, a local web server, and the target application.

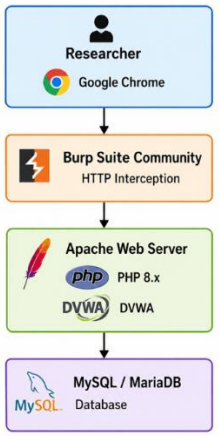

Figure 1. The experimental architecture

The researcher interacts with the DVWA application using Google Chrome. HTTP requests and responses are intercepted using Burp Suite Community Edition to analyze request parameters, cookies, and transmitted payloads. The target application is hosted locally using Apache and PHP within the XAMPP environment, while MySQL/MariaDB serves as the backend database management system.

C. Experimental Environment

The experiment was conducted using the hardware and software configuration presented in Table I.

Table 1 Experimental Environment

Component	Specification
Operating System	Windows 11 Pro 64-bit
Processor	Intel Core i5 / AMD Ryzen 5 (or equivalent)
Memory	8 GB RAM
Web Server	Apache 2.4 (XAMPP)
Database	MySQL / MariaDB
Programming Language	PHP 8.x
Target Application	Damn Vulnerable Web Application (DVWA)
Browser	Google Chrome
Penetration Testing Tool	Burp Suite Community Edition
Supporting Tool	Browser Developer Tools

The experimental environment was isolated from external networks to eliminate unintended security risks and ensure compliance with ethical penetration testing practices.

D. Research Materials

The primary research object was the Damn Vulnerable Web Application (DVWA), an intentionally vulnerable web application developed for cybersecurity education and penetration testing training.

DVWA contains numerous web application vulnerabilities, including:

1. SQL Injection
2. Blind SQL Injection
3. Cross-Site Scripting (XSS)
4. Command Injection
5. File Inclusion
6. Cross-Site Request Forgery (CSRF)
7. File Upload Vulnerabilities

This study specifically focuses on SQL Injection because it remains one of the most critical vulnerabilities identified by OWASP Top 10 (2021) and MITRE CWE-89.

The application security level was configured to **Low** to demonstrate insecure coding practices and facilitate controlled vulnerability exploitation.

E. Penetration Testing Methodology

The penetration testing procedure consists of six sequential phases.

1) Environment Preparation

The first phase involved installing DVWA within the XAMPP environment and configuring Apache, PHP, and MySQL services. Database initialization was performed to ensure that all application components functioned correctly before security testing commenced.

2) Information Gathering

Reconnaissance activities were performed to identify application components that interact with backend databases.

The following information was collected:

1. Login pages
2. HTML forms
3. URL parameters
4. Cookies
5. HTTP Headers
6. Session identifiers
7. HTTP GET requests
8. HTTP POST requests

Burp Suite Proxy and Browser Developer Tools were used to intercept and analyze all HTTP communication.

3) Vulnerability Identification

Potential SQL Injection entry points were identified by injecting SQL special characters into user-controlled input fields.

Initial payloads included:

```
'
"
'--
'#
```

Application responses were analyzed to determine whether database errors or abnormal behaviors occurred.

Indicators of SQL Injection vulnerability include:

1. SQL syntax errors
2. Database error messages
3. Unexpected application behavior
4. Authentication anomalies

4) SQL Injection Exploitation

After confirming vulnerable parameters, additional SQL Injection payloads were executed to evaluate exploitation severity.

Representative payloads included:

```
' OR '1'='1' --
' UNION SELECT user,password FROM users--
' UNION SELECT database(),user()--
```

The objective of this phase was to evaluate:

1. Authentication bypass
2. Information disclosure
3. Database enumeration
4. Unauthorized data access

Each payload was executed manually to observe application behavior under controlled conditions.

5) Risk Assessment

Identified vulnerabilities were evaluated using internationally recognized cybersecurity standards.

Security evaluation was performed based on:

1. OWASP Top 10 (2021)
2. CWE-89
3. CVSS v3.1

The following evaluation criteria were considered:

1. Attack Complexity

2. Privileges Required
3. User Interaction
4. Confidentiality Impact
5. Integrity Impact
6. Availability Impact

The final severity level was determined according to the CVSS Base Score.

6) Mitigation Recommendation

Based on the experimental findings, mitigation strategies were proposed by referring to secure software development best practices.

Recommended controls include:

1. Parameterized Queries
2. Prepared Statements
3. Server-side Input Validation
4. Input Sanitization
5. Least Privilege Principle
6. Secure Session Management
7. Web Application Firewall (WAF)
8. Secure Software Development Life Cycle (SSDLC)
9. DevSecOps Integration

F. Research Workflow

The overall research workflow is illustrated as follows.

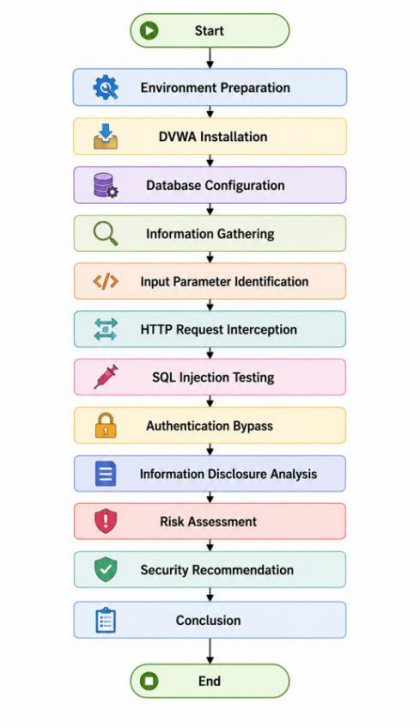

Figure 2. Research workflow

G. Data Analysis

The experimental results were analyzed using descriptive qualitative analysis supported by standardized cybersecurity evaluation frameworks.

The analysis focused on four primary aspects:

1. Identification of vulnerable application components.
2. Evaluation of SQL Injection payload effectiveness.
3. Assessment of security impacts according to OWASP Top 10 (2021), CWE-89, and CVSS v3.1.
4. Development of mitigation strategies based on secure coding principles and secure software development frameworks.

To improve the reliability of the findings, each successful exploitation attempt was verified through repeated testing and observation of application responses. The experimental results were subsequently compared with findings reported in previous studies to ensure consistency and strengthen the validity of the research.

IV. RESULTS AND DISCUSSION

A. Vulnerability Identification

The penetration testing experiment was conducted on the Damn Vulnerable Web Application (DVWA) configured at the **Low Security** level. Initial testing focused on identifying application components that directly interacted with the backend database. Manual inspection using Burp Suite Community Edition and Browser Developer Tools confirmed that several user-controlled parameters were transmitted to the server without adequate validation or sanitization.

To verify the presence of SQL Injection vulnerabilities, a series of preliminary payloads containing SQL special characters and logical operators were submitted through the application's input fields. The application responses were then analyzed to determine whether SQL statements were executed improperly.

Table 2 Initial Sql Injection Test Results

Payload	Purpose	Application Response	Status
'	Detect SQL syntax error	SQL syntax error displayed	Vulnerable
''	Detect malformed query	SQL syntax error displayed	Vulnerable
'--	Comment injection	SQL query interrupted	Vulnerable
' OR '1'='1'	Authentication bypass	Login successful	Vulnerable
' UNION SELECT NULL--	UNION testing	SQL response generated	Vulnerable

The experimental results indicate that all tested payloads successfully influenced SQL query execution. The appearance of SQL syntax errors confirms that user input was concatenated directly into SQL statements without adequate server-side validation. According to the OWASP Web Security Testing Guide, this behavior strongly indicates the presence of SQL Injection vulnerabilities.

Furthermore, database error messages exposed valuable information regarding database structures and query execution, potentially assisting attackers in constructing more sophisticated exploitation techniques.

B. Authentication Bypass Analysis

Authentication bypass represents one of the most severe consequences of SQL Injection vulnerabilities because it enables attackers to access protected resources without possessing valid credentials.

The following payload was submitted through the login form:

' OR '1'='1' --

The injected payload modified the original SQL statement into a logically valid expression:

SELECT *
FROM users
WHERE username=''
OR '1'='1';

Since the logical expression '1'='1' always evaluates to TRUE, the database returned at least one valid record, causing the application to authenticate the attacker successfully.

The successful authentication bypass demonstrates that the application failed to distinguish executable SQL commands from user-supplied input. This weakness primarily resulted from insecure query construction through dynamic string concatenation rather than parameterized SQL execution.

Figure 3 presents the successful authentication bypass obtained during the penetration testing experiment.

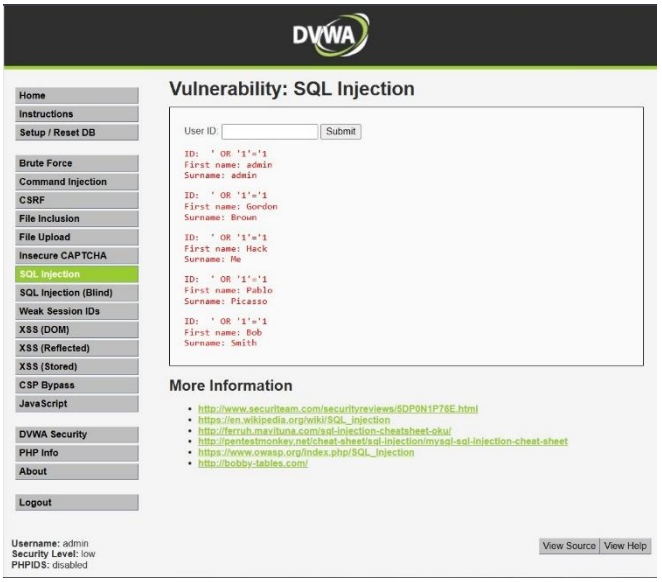

Figure 3 Presents the successful authentication bypass obtained during the penetration testing experiment.

C. Database Information Disclosure

Following successful authentication bypass, additional SQL Injection payloads were executed to evaluate whether confidential information stored within the database could be retrieved.

The following UNION-based payload was successfully executed:

' UNION SELECT user,password FROM users--

The experiment demonstrated that database records containing usernames and password hashes could be extracted without requiring legitimate database credentials.

The successful execution of UNION-based SQL Injection indicates that the application failed to restrict unauthorized SQL query manipulation.

Potential consequences include:

1. Unauthorized disclosure of sensitive information.
2. Credential harvesting.
3. Identity theft.
4. Privilege escalation.
5. Lateral movement within organizational networks.

These findings are consistent with the CWE-89 classification, which identifies SQL Injection as one of the most critical software security weaknesses.

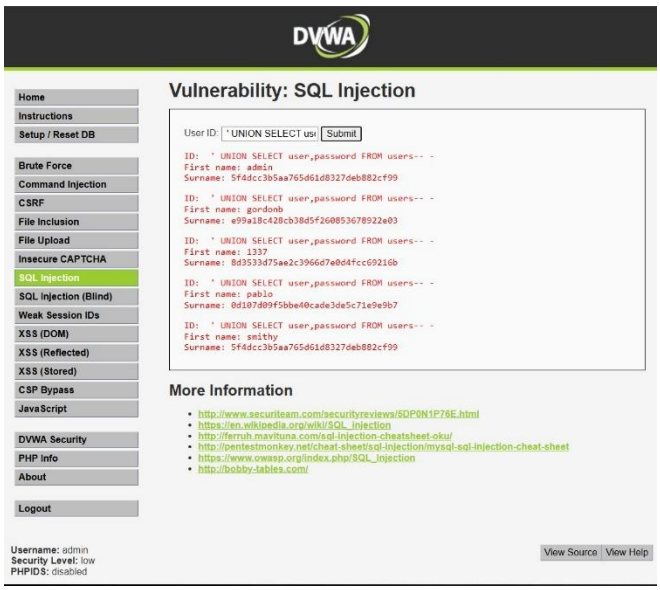

Figure 4 Most critical software security weaknesses

D. Security Weakness Analysis

The penetration testing experiment revealed several fundamental weaknesses within the application's implementation.

1. Insufficient Input Validation

The application accepted user-supplied input without validating special SQL characters before processing database queries. Consequently, malicious SQL statements were executed directly by the database management system.

2. Dynamic SQL Query Construction

The vulnerable application constructed SQL statements through direct string concatenation.

Example:

```
SELECT *
FROM users
WHERE username='$username'
AND password='$password';
```

This implementation allows attackers to modify SQL logic by injecting malicious input into user-controlled parameters.

3. Absence of Prepared Statements

Prepared Statements separate executable SQL commands from user input, preventing malicious payloads from being interpreted as SQL syntax.

The absence of parameterized queries significantly increased the application's susceptibility to SQL Injection attacks.

4. Excessive Database Privileges

The database account possessed permissions exceeding operational requirements.

In production environments, similar privilege configurations may enable attackers to:

1. Delete database contents.
2. Modify organizational records.
3. Create administrative accounts.
4. Execute privileged SQL commands.

Implementing the Principle of Least Privilege (PoLP) is therefore essential for minimizing exploitation impact.

E. Risk Assessment

The identified vulnerability was evaluated according to OWASP Top 10 (2021), MITRE CWE-89, and CVSS Version 3.1.

Table 3 Security Risk Assessment

Evaluation Criteria	Result
OWASP Category	A03:2021 – Injection
CWE Classification	CWE-89
Attack Vector	Network
Attack Complexity	Low
Privileges Required	None
User Interaction	None
Confidentiality Impact	High
Integrity Impact	High
Availability Impact	High
Estimated CVSS Base Score	9.8 (Critical)

The vulnerability received a **Critical** severity rating because successful exploitation required minimal effort while potentially compromising all three core principles of information security: confidentiality, integrity, and availability.

F. Comparison with Previous Studies

To evaluate the significance of the findings, the experimental results were compared with representative studies in SQL Injection research.

Table 4 Comparison With Previous Studies

Research	Method	Main Findings	Comparison with This Study
Halfond et al.	Classification	SQLi taxonomy	This study experimentally validates

			authentication bypass.
OWASP (2021)	Security guideline	Injection remains critical	Experimental findings support OWASP recommendations.
Alwan & Younis	Secure coding	Prepared Statements prevent SQLi	Experimental evidence confirms their effectiveness.
Li et al.	Machine learning	Intelligent SQLi detection	This study focuses on practical exploitation rather than automated detection.
This Research	Manual Penetration Testing	Authentication bypass, information disclosure, and OWASP/CVSS-based risk assessment	Provides integrated practical validation and mitigation recommendations.

Compared with previous research, this study emphasizes practical experimentation through manual penetration testing and demonstrates how insecure coding practices directly influence web application security.

G. Discussion

The experimental findings confirm that SQL Injection remains a significant threat to web application security despite the widespread availability of secure software development frameworks.

The success of authentication bypass demonstrates that insecure query construction continues to be one of the primary causes of SQL Injection vulnerabilities. Applications relying on direct string concatenation remain highly susceptible to malicious input manipulation, particularly when server-side validation and parameterized queries are absent.

From a software engineering perspective, these findings reinforce the importance of integrating secure coding practices into every phase of the Secure Software Development Life Cycle (SSDLC). Security should not be treated solely as a post-development testing activity but should instead be embedded throughout system design, implementation, testing, deployment, and maintenance.

Furthermore, the results support DevSecOps principles by demonstrating that continuous penetration testing can identify exploitable vulnerabilities before deployment into production environments. Integrating security testing into continuous integration and continuous deployment (CI/CD) pipelines enables organizations to reduce remediation costs while improving software quality and organizational resilience against cyber threats.

Although the experiments were conducted using DVWA, the identified vulnerabilities closely resemble SQL Injection weaknesses that continue to be discovered in real-world web applications. Consequently, the findings remain valuable for cybersecurity education, penetration testing training, and secure software development.

H. Threats to Validity

Several limitations should be considered when interpreting the findings of this research.

First, the experiment was conducted exclusively using DVWA, which is intentionally designed as a vulnerable educational platform. Therefore, the observed vulnerabilities may not fully represent the security characteristics of modern production web applications.

Second, this study focused solely on manual penetration testing techniques. Automated vulnerability assessment tools such as SQLMap, OWASP ZAP, and Burp Suite Professional were not included in the experimental evaluation.

Third, only SQL Injection vulnerabilities were investigated. Other critical web application vulnerabilities, including Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Broken Access Control, and Server-Side Request Forgery (SSRF), were outside the scope of this research.

Despite these limitations, the controlled experimental environment provides reliable and reproducible evidence regarding SQL Injection exploitation and secure coding mitigation strategies.

I. Practical Implications

The findings of this research provide practical recommendations for software developers, cybersecurity practitioners, and higher education institutions.

For software developers, the results emphasize the necessity of implementing parameterized queries, prepared statements, input validation, secure session management, and least-privilege database configurations.

For organizations, periodic penetration testing should be incorporated into organizational cybersecurity governance and DevSecOps practices to ensure that exploitable vulnerabilities are identified before deployment.

For educational institutions, DVWA remains an effective learning platform for demonstrating web application vulnerabilities within a controlled and legally compliant environment.

V. CONCLUSION

This study investigated SQL Injection vulnerabilities within the Damn Vulnerable Web Application (DVWA) through a structured manual penetration testing approach conducted in a controlled laboratory environment. The experimental results demonstrated that the application configured at the **Low Security** level was highly susceptible to SQL Injection attacks due to insufficient server-side input validation, insecure dynamic

SQL query construction, and the absence of parameterized queries and prepared statements.

The penetration testing activities successfully identified exploitable SQL Injection vulnerabilities that enabled authentication bypass and unauthorized disclosure of sensitive database information. The findings confirm that improper handling of user-supplied input continues to represent a critical security weakness capable of compromising the confidentiality, integrity, and availability of web-based information systems. Risk assessment based on **OWASP Top 10 (2021)**, **CWE-89**, and **CVSS v3.1** classified the identified vulnerability as **Critical**, emphasizing the necessity of integrating secure coding principles throughout the software development lifecycle.

The findings also address the research questions proposed in this study. The experiment confirmed that vulnerable input parameters within DVWA could be exploited using manual SQL Injection payloads, resulting in successful authentication bypass and information disclosure. Furthermore, the study demonstrated that implementing parameterized queries, prepared statements, comprehensive server-side input validation, secure session management, least-privilege database configurations, and Web Application Firewall (WAF) technologies represents an effective strategy for mitigating SQL Injection vulnerabilities.

From a scientific perspective, this research contributes by presenting an integrated penetration testing framework that combines manual SQL Injection exploitation, authentication bypass verification, vulnerability assessment, and security risk evaluation using internationally recognized standards, including OWASP Top 10 (2021), CWE-89, and CVSS v3.1. Unlike studies that primarily focus on theoretical discussions or automated detection techniques, this research provides practical experimental evidence illustrating how insecure coding practices directly influence web application security.

From a practical perspective, the proposed methodology can serve as a reference for software developers, cybersecurity practitioners, educators, and information system administrators seeking to evaluate and improve the security posture of web applications. Furthermore, integrating penetration testing into the Secure Software Development Life Cycle (SSDLC) and DevSecOps practices enables organizations to identify security weaknesses during software development, thereby reducing remediation costs and strengthening organizational cybersecurity resilience.

Although this study provides valuable practical insights, several limitations should be acknowledged. The experiments were conducted exclusively using the Damn Vulnerable Web Application (DVWA) configured at the Low Security level, which represents a controlled educational environment rather than a production web application. In addition, the study focused solely on manual penetration testing and SQL Injection vulnerabilities without evaluating other common web security threats or automated security assessment tools.

Future research should extend this work by investigating SQL Injection attacks across multiple DVWA security levels, including Medium, High, and Impossible, to evaluate the effectiveness of progressively stronger defensive mechanisms. Comparative analyses involving automated penetration testing tools such as SQLMap, OWASP ZAP, and Burp Suite Professional would provide deeper insights into the strengths and limitations of manual and automated testing approaches. Future studies may also investigate additional web application vulnerabilities, including Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Server-Side Request Forgery (SSRF), Broken Access Control, and Remote Code Execution (RCE). Furthermore, integrating artificial intelligence and machine learning techniques into vulnerability detection, as well as incorporating penetration testing into modern DevSecOps pipelines, represents a promising direction for enhancing secure software development and organizational cybersecurity.

REFERENCES

[1] OWASP Foundation, *OWASP Top 10: The Ten Most Critical Web Application Security Risks*, 2021.
[2] OWASP Foundation, *OWASP Web Security Testing Guide (WSTG) Version 4.2*, 2021.
[3] OWASP Foundation, *SQL Injection Prevention Cheat Sheet*, 2023.
[4] OWASP Foundation, *Application Security Verification Standard (ASVS) Version 4.0.3*, 2021.
[5] MITRE Corporation, *CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*, 2024.
[6] FIRST, *Common Vulnerability Scoring System (CVSS) Version 3.1 Specification*, 2019.
[7] National Institute of Standards and Technology (NIST), *Secure Software Development Framework (SSDF) Version 1.1*, NIST SP 800-218, 2022.
[8] National Institute of Standards and Technology (NIST), *Security and Privacy Controls for Information Systems and Organizations*, NIST SP 800-53 Rev. 5, 2020.
[9] PortSwigger Ltd., *Burp Suite Documentation*, 2024.
[10] PortSwigger Ltd., *Web Security Academy – SQL Injection*, 2024.
[11] DVWA Project Team, *Damn Vulnerable Web Application Documentation*, GitHub, 2024.
[12] A. Halfond, J. Viegas, and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," *Proceedings of the International Symposium on Secure Software Engineering*, 2006.
[13] A. K. Jain and B. B. Gupta, "SQL Injection Detection and Prevention Techniques: A Survey," *Journal of Information Security and Applications*, vol. 56, 2021.
[14] M. Almseidin, M. Alzubi, S. Kovacs, and M. Alkasassbeh, "Evaluation of SQL Injection Detection Techniques for Web Applications," *IEEE Access*, vol. 10, pp. 48731–48745, 2022.
[15] B. B. Gupta and A. Goyal, "Recent Advances in Web Application Security Against Injection Attacks," *Future Generation Computer Systems*, vol. 131, pp. 190–205, 2022.

[16] N. Antunes and M. Vieira, "Comparing the Effectiveness of Penetration Testing and Static Code Analysis in Web Security," *Information and Software Technology*, vol. 145, 2022.
[17] R. Kumar, S. Tanwar, and N. Kumar, "Secure Web Applications Through Penetration Testing Approaches," *IEEE Access*, vol. 11, pp. 21150–21170, 2023.
[18] S. Ali, M. Khan, and H. Abbas, "Analysis of SQL Injection Vulnerabilities Using Automated Security Testing," *Computers & Security*, vol. 128, 2023.
[19] Y. Li, H. Zhang, and J. Wang, "Machine Learning-Based Detection of SQL Injection Attacks in Modern Web Applications," *IEEE Access*, vol. 11, pp. 74532–74546, 2023.
[20] M. Alenezi and A. Alshammari, "A Comparative Analysis of SQL Injection Prevention Mechanisms," *Journal of King Saud University – Computer and Information Sciences*, vol. 35, no. 7, 2023.
[21] A. Alwan and M. Younis, "Secure Coding Practices for Web Application Security," *International Journal of Information Security*, vol. 22, no. 2, pp. 221–238, 2023.
[22] B. Schneier, *Secrets and Lies: Digital Security in a Networked World*. Wiley, 2015.
[23] R. S. Pressman and B. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. McGraw-Hill, 2020.
[24] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2016.
[25] W. Stallings, *Network Security Essentials: Applications and Standards*, 7th ed. Pearson, 2020.
[26] W. Stallings, *Effective Cybersecurity*. Addison-Wesley, 2018.
[27] CERT Coordination Center, *CERT Secure Coding Standards*, Carnegie Mellon University, 2022.
[28] Open Worldwide Application Security Project, *OWASP DevSecOps Guideline*, 2023.
[29] NIST, *National Vulnerability Database (NVD)*, 2024.
[30] OpenSSF, *Secure Software Development Fundamentals*, 2023.
[31] M. Howard and D. LeBlanc, *Writing Secure Code*, 2nd ed. Microsoft Press, 2003.
[32] S. McConnell, *Code Complete*, 2nd ed. Microsoft Press, 2004.
[33] C. Anley, J. Heasman, F. Lindner, and G. Richarte, *The Shellcoder's Handbook*, 2nd ed. Wiley, 2007.
[34] B. B. Gupta, *Handbook of Computer Networks and Cyber Security*, Springer, 2020.
[35] IEEE Computer Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK v4)*, 2024.
[36] ENISA, *Threat Landscape Report*, European Union Agency for Cybersecurity, 2024.
[37] CISA, *Secure by Design: Shifting the Balance of Cybersecurity Risk*, 2024.
[38] OWASP Foundation, *OWASP API Security Top 10*, 2023.
[39] MITRE ATT&CK Team, *MITRE ATT&CK Enterprise Matrix*, 2024.
[40] ISO/IEC 27001:2022, *Information Security, Cybersecurity and Privacy Protection — Information Security Management Systems — Requirements*, ISO, 2022.