Report
DVWA
Lab Machine:
SQL Injection.

Francisco José Morante Lozada
4Geeks Student

Index

About This Report:

- This document demonstrates security testing performed in a controlled lab environment using intentionally vulnerable machines.
- All testing conducted on dedicated lab systems
- No real systems or data were compromised
- "Gaining machine control" refers to educational exercises
- Primary goal: Perform an SQL Injection on the target machine
- These exercises help understand attacker methodologies to build better defenses.

1st Introduction

This report details the discovery and validation of a critical SQL Injection (SQLi) vulnerability within the public-facing product search feature of ExampleWebApp. The assessment was conducted as part of a scheduled security review with the following agreed scope:

In-Scope Target:

http://192.168.1.219/dvwa/

In-Scope Test Type: Focused testing for injection vulnerabilities, specifically SQLi, against user-controllable inputs.

Out-of-Scope: Testing for other vulnerability classes (e.g., XSS, CSRF) was not the primary objective of this engagement.

The vulnerability allows an unauthenticated attacker to extract sensitive data from the application's backend database.

2nd Incident Description

In-Band SQL Injection (Union-Based) in User Lookup Function

Location: The web application's user lookup or search functionality (as indicated by a "User ID" input field and a "[Submit]" button).

The application features a form where users can query information by a numeric "User ID." The underlying code directly concatenates user-supplied input into an SQL SELECT statement without any sanitization or parameterization.

3rd Reproduction Process

An attacker can exploit this vulnerability without any specialized tools, using only a web browser.

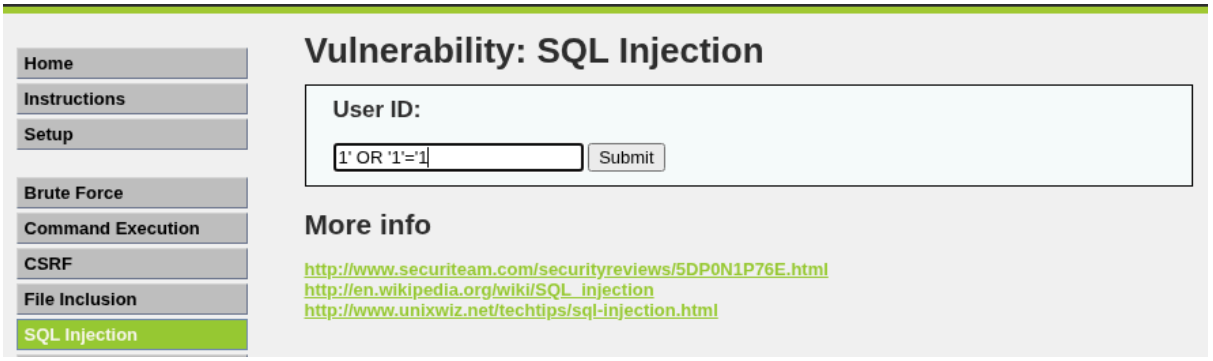
Navigate to the product search page:

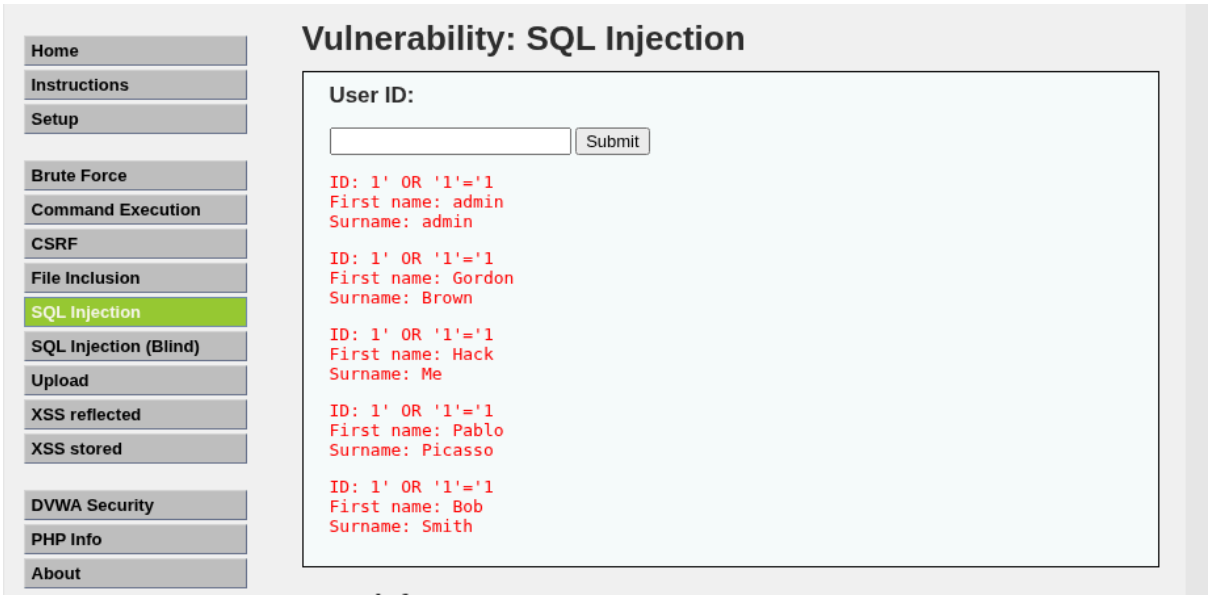
http://192.168.1.219/dvwa/vulnerabilities/sqli/

In the search box, enter the following payload:

1' OR '1'='1

Submit the search. Instead of product results, the application's response page will display a list of system usernames and password hashes from the users table.





4th Incident Impact

Confidentiality	Critical Impact. Attackers can exfiltrate all data stored in the connected database, including user credentials (passwords), personal identifiable information (PII), financial records, and proprietary business data.
Integrity	High Impact. While demonstrated as a data theft vector, the same flaw could potentially allow modification or deletion of database records (INSERT, UPDATE, DELETE), leading to data corruption or loss.
Availability	Medium Impact. Malformed SQL payloads can cause database errors and denial-of-service, making the product search feature unavailable.
Business Risk	This flaw constitutes a direct breach of data protection regulations (e.g., GDPR, CCPA), could lead to significant reputational damage, and provides a direct path for full system compromise.

5th Recommendations

Immediate action is required to remediate this vulnerability.

Immediate Mitigation (Within 24 hours):

Deploy a Web Application Firewall (WAF) rule to block SQL meta-characters (e.g., single quotes ', comment sequences --) in the search parameter. This is a temporary, tactical fix.

Primary Remediation (Within 7 days):

Fix the Root Cause: Refactor the vulnerable code to use parameterized queries (prepared statements). This ensures user input is treated strictly as data, not executable SQL code.

Example Code Fix:

python
# VULNERABLE CODE (Current State)
query = "SELECT * FROM products WHERE name LIKE '%" + user_input + "%'"
cursor.execute(query)

# REMEDIATED CODE (Using Parameterization)
query = "SELECT * FROM products WHERE name LIKE %s"
cursor.execute(query, ('%' + user_input + '%',))
Long-Term Prevention:

Implement mandatory secure coding training for developers focused on injection flaws.

Integrate Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) tools into the CI/CD pipeline to catch similar issues early.

6th Conclusion

The identified SQL Injection vulnerability poses a critical risk to the security and integrity of the ExampleWebApp platform and its data. Exploitation is trivial and can lead to a full breach of the application database. It is recommended that the development team prioritize and implement the parameterized query fix as the permanent solution. A follow-up test is advised post-remediation to confirm the vulnerability is fully resolved.

7th Legal and Compliance Consequences:

General Data Protection Regulation (GDPR)

Article 32 Violations, Security of Processing:

- No Encryption of personal data (Passwords in MD5 and Plaintext at the web)
- No integrity protection (SQL Injection)
- No confidentiality assurance.

Article 33 Violations, Personal Data Breach:

- Mandatory 72-hour notification to supervisory authority
- Direct notification to affected data subjects

Article 5 Violation, Principles Relating to Processing. Everyone can access the server, so the Data Base is vulnerable:

- Purpose limitation (Data accessed beyond intended purpose)
- Data minimization (Data exposure through SQL Injection)
- Accuracy (Data integrity compromised)
- Storage limitation (Data retention exposed)

ISO/IEC 27001:2022 Violations

Annex A.5.7, Threat Intelligence:

- Using outdated software with Known CVEs.
- Maintain awareness of threats and vulnerabilities.

Annex A.8.28, Secure Coding:

- SQL Injection.
- No secure development implemented

8th Sources

4Geeks: spain-cs-pt-11/project/incident-report-for-sql-injection-exercise-project

OWASP Top Ten: https://owasp.org/Top10/2021/A03_2021-Injection/

cvedetails.com: Preventions.