Lab 1: Algorithm and Code Analysis of Website or Application (Using DVWA)

Objective

To demonstrate how SQL Injection and Cross-Site Scripting (XSS) attacks work and how to analyze vulnerable code logic in insecure web applications using DVWA (Damn Vulnerable Web Application).

Software/Tools Required

DVWA (Damn Vulnerable Web Application)

XAMPP

Web Browser: Chrome or Firefox

Theory

SQL Injection (SQLi)

SQL Injection is a web vulnerability that allows an attacker to modify or inject SQL queries in a web application's database layer.
It typically happens when:

User input is directly inserted into SQL queries.

No input validation or escaping is performed.

Cross-Site Scripting (XSS)

XSS allows an attacker to **inject malicious JavaScript code** into a web page viewed by other users.
It occurs when:

Web applications display unsanitized user input in the browser.

The browser executes the injected script as part of the page.

This can lead to:

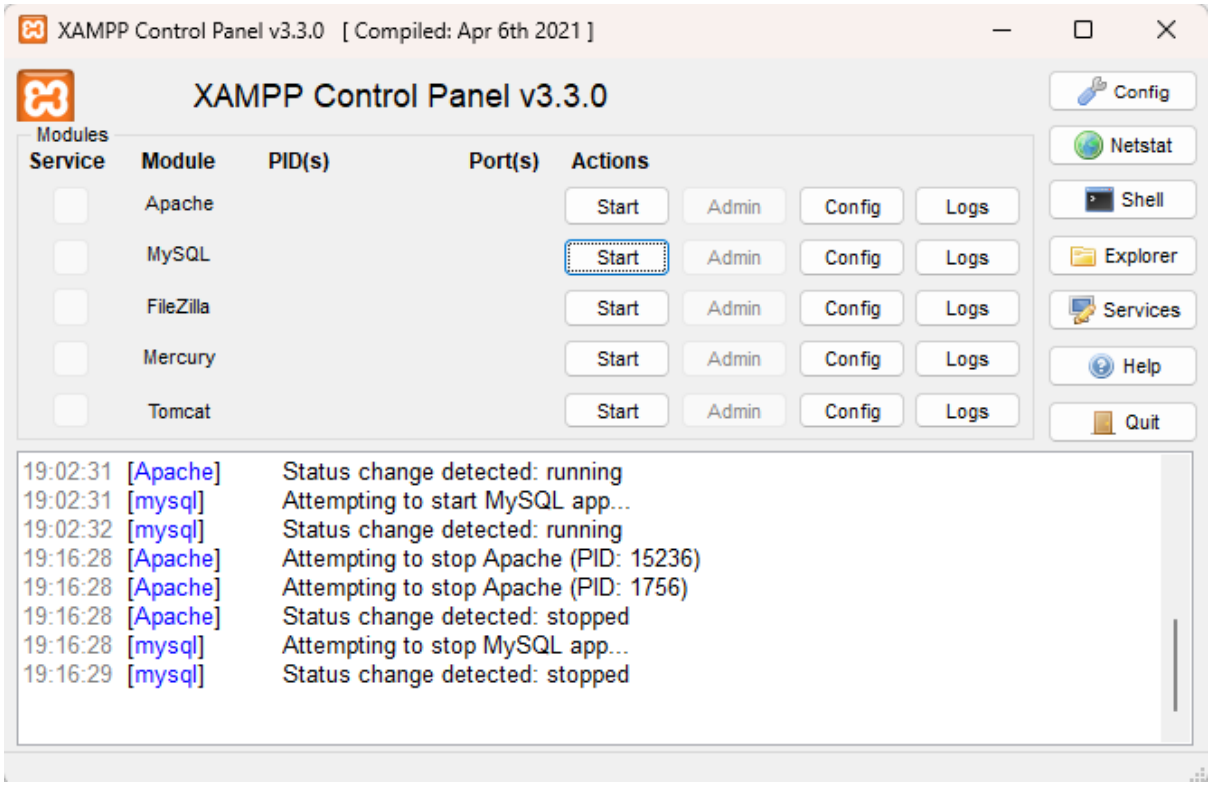
Cookie theft

Session hijacking

Redirection to malicious sites

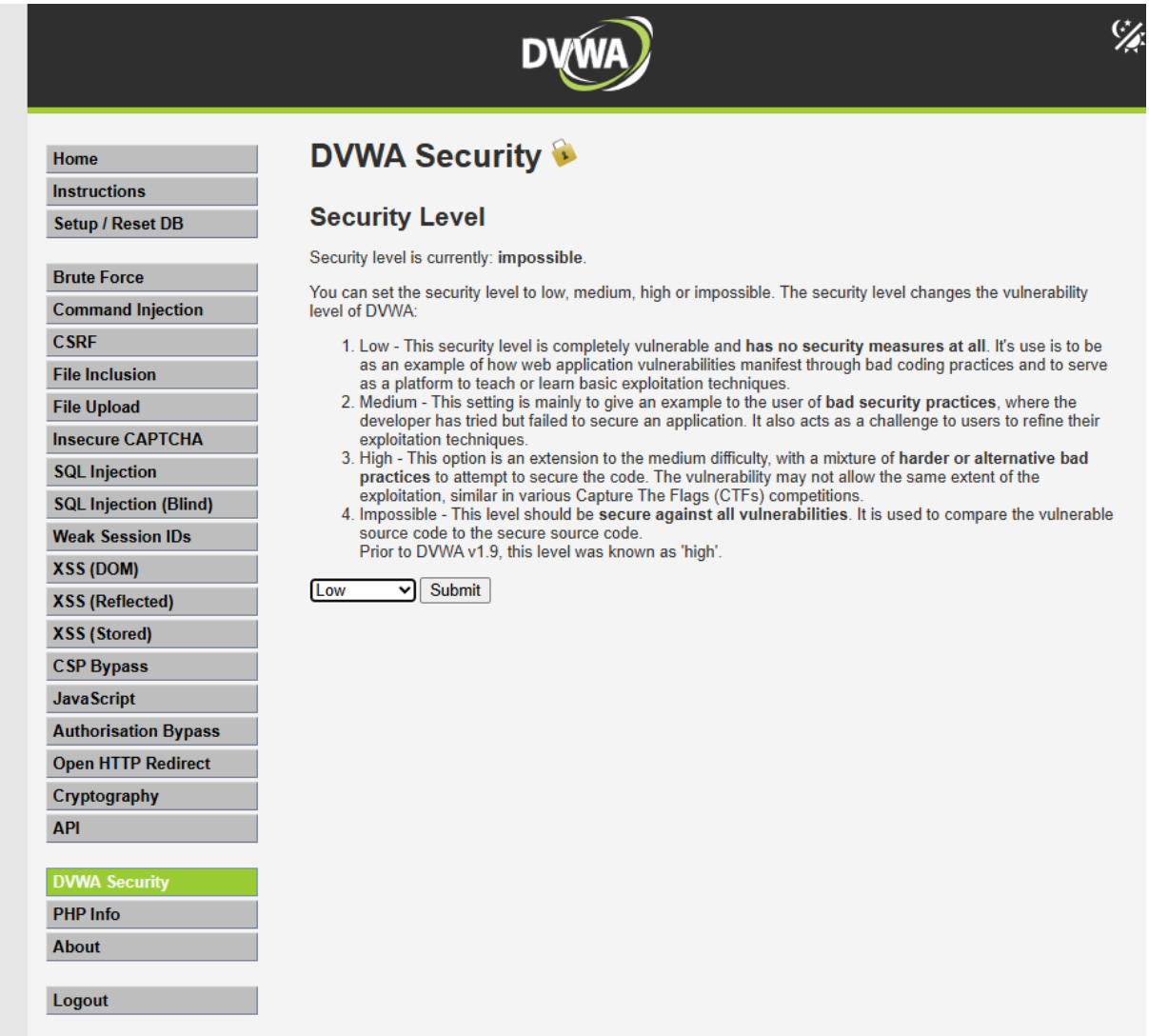
SQL Injection in DVWA

Step 1: Open Xampp and run apache and mysql



Step 2: Login to DVWA (admin / password)

Step 3: Set security to Low and submit



Step 4: Go to SQL Injection option

Step 5: Go to View Source option

Vulnerability: SQL Injection

User ID: 1 Submit

```
ID: 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/

View Source | View Help

Step 6: Analyse the code to detect the vulnerability

SQL Injection Source

vulnerabilities/sqli/source/low.php

```php
<?php

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

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            // 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"]);
            break;
        case SQLITE:
            global $sqlite_db_connection;

            #$sqlite_db_connection = new SQLite3($_DVWA['SQLITE_DB']);
            #$sqlite_db_connection->enableExceptions(true);

            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
            #print $query;
            try {
                $results = $sqlite_db_connection->query($query);
            } catch (Exception $e) {
                echo 'Caught exception: ' . $e->getMessage();
                exit();
            }

            if ($results) {
                while ($row = $results->fetchArray()) {
                    // Get values
                    $first = $row["first_name"];
                    $last  = $row["last_name"];

                    // Feedback for end user
                    echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
                }
            } else {
                echo "Error in fetch ".$sqlite_db->lastErrorMsg();
            }
            break;
    }
}

?>
```

Problem 1: Unsafe User Input

if(isset($_REQUEST['Submit'])) {

    // Get input

    $id = $_REQUEST['id'];

- $_REQUEST['id'] takes user input without validation.
- Accepts anything, including special characters and SQL code like: 1' OR '1'='1

Problem 2: Query is Built Using Raw Input

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

- User input is directly inserted into the SQL query as a string.
- This allows a user to change the logic of the query by adding SQL keywords.

Why It's Vulnerable?

Let’s say the attacker provides this as input in the browser: ?id=1' OR '1'='1

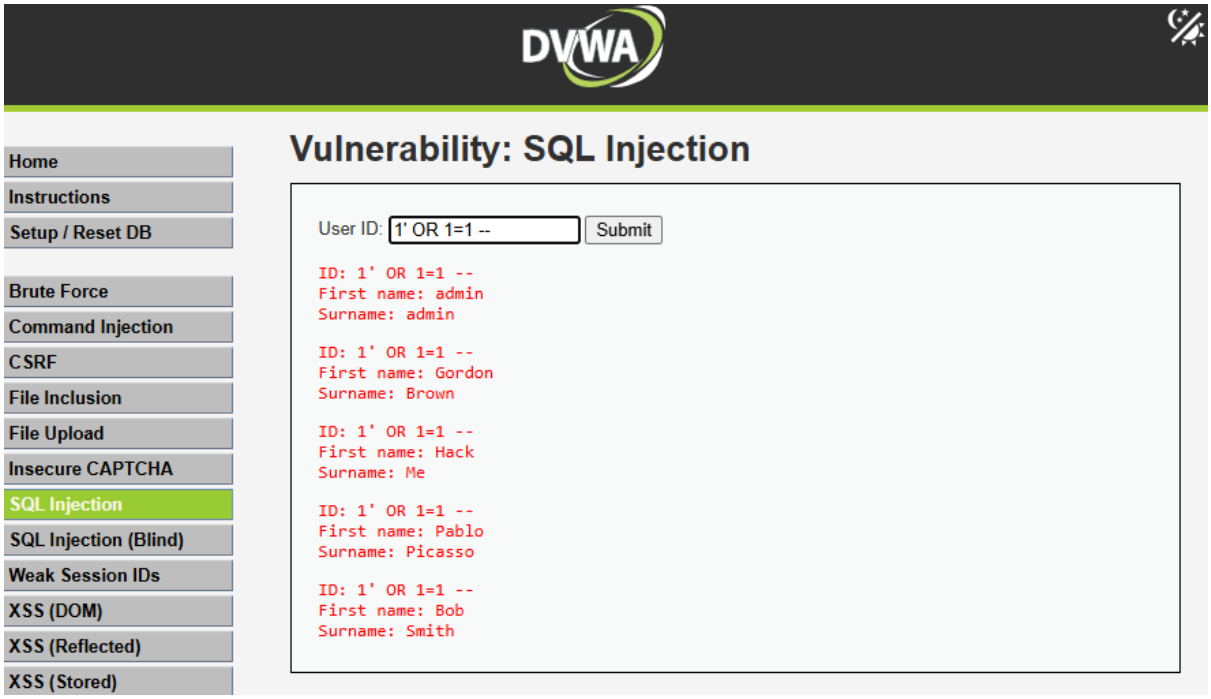
The query becomes:

SELECT first_name, last_name FROM users WHERE user_id = '1' OR '1'='1';

'1'='1' is always true, so the database returns all rows, bypassing the intention of fetching just one user.

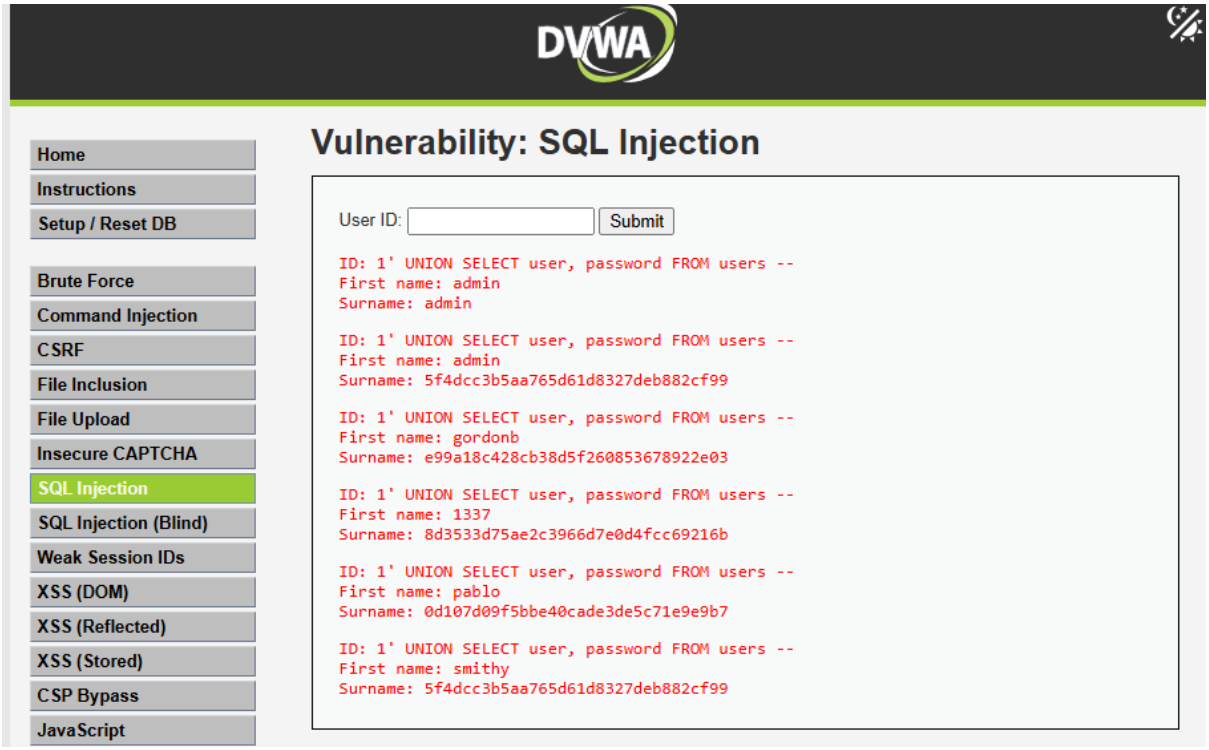
Step 7: Perform SQL injection (to show first name and surname from the database)

Payload : 1' OR 1=1 – (don't forget to add an extra space at the end of the payload)

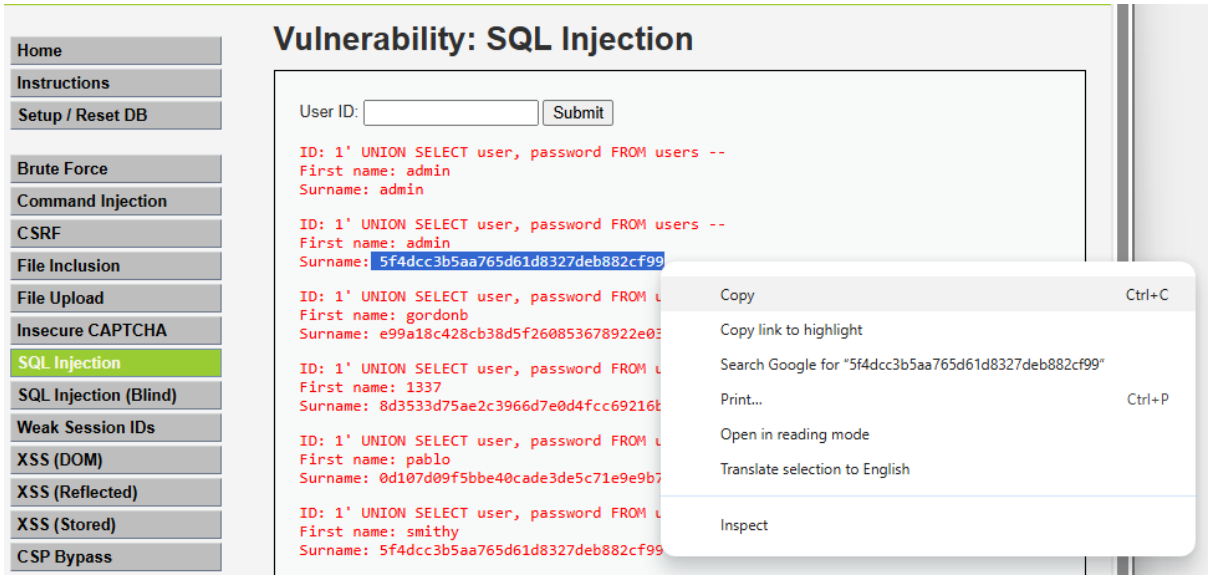


Step 8: Perform SQL injection (to show user id and password from the database)

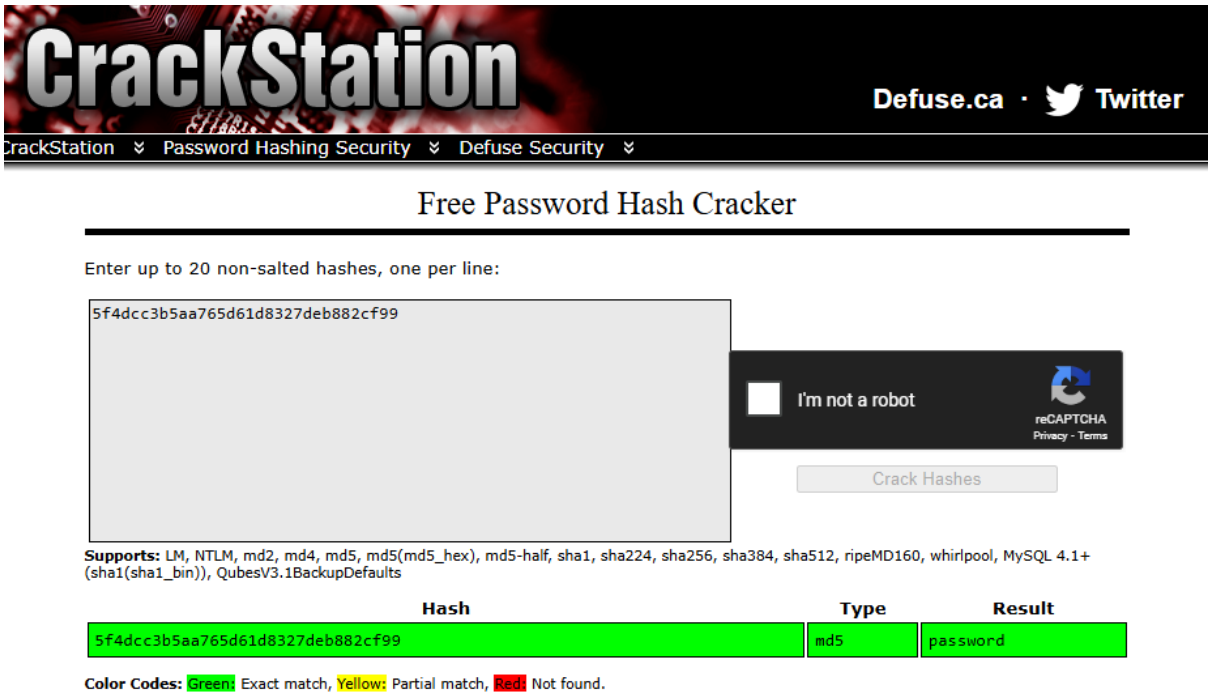
Payload : 1' UNION SELECT user, password FROM users -- (don't forget to add an extra space at the end of the payload)



Step 9: Copy the hashed password



Step 10: Go to https://crackstation.net to crack the password

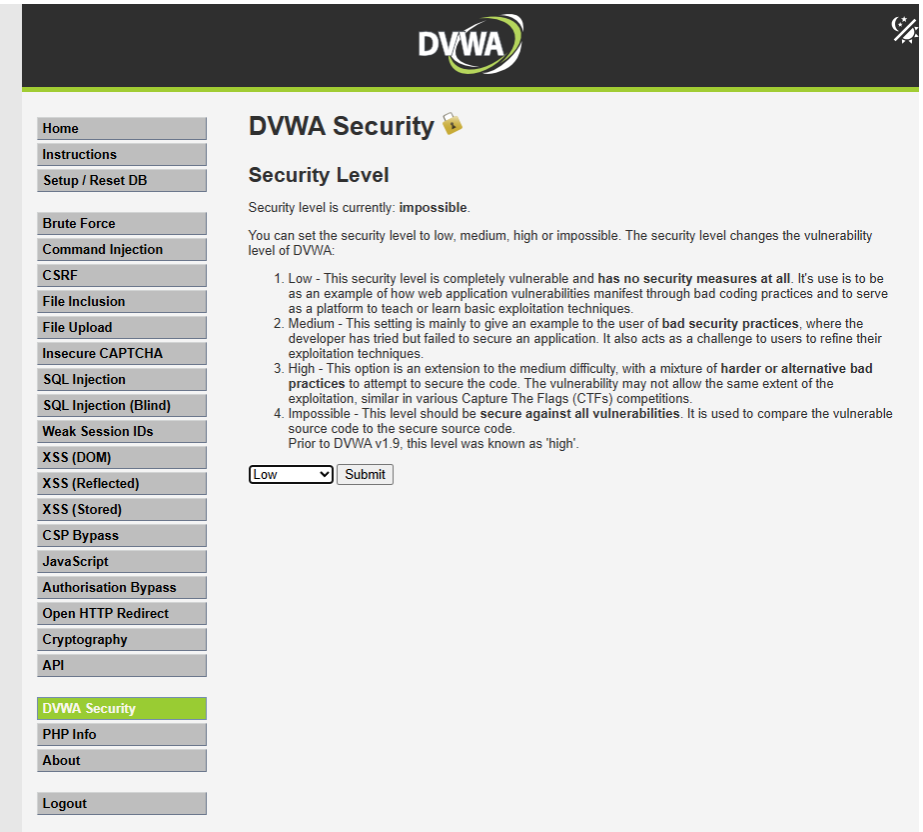


RESULT:

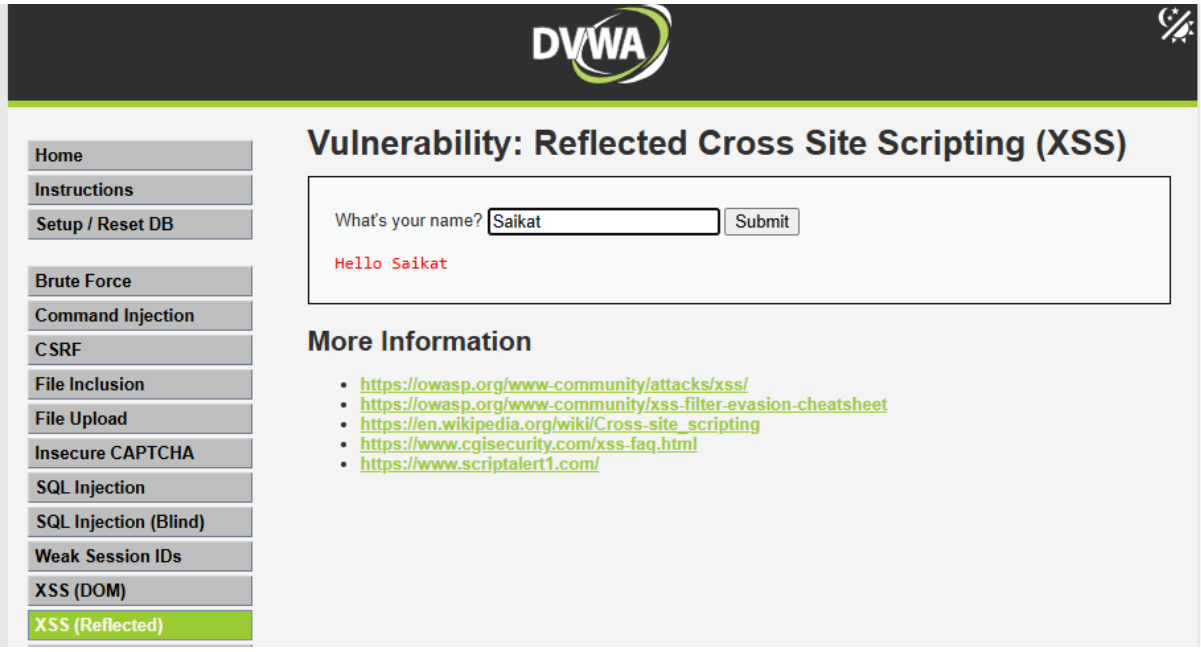
Password is : password

Analysing the vulnerable code of XSS (Cross Site Scripting) in DVWA:

Step 1: Set security to Low and submit

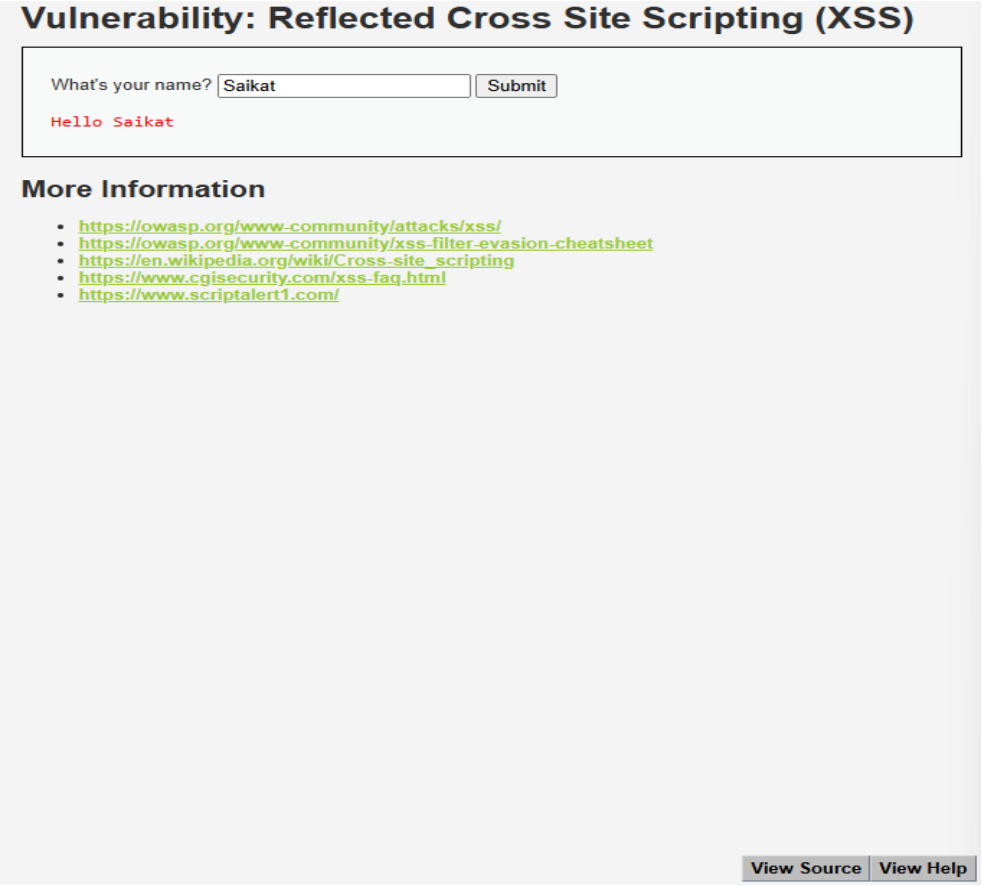


Step 2: Go to XSS (Reflected)



- **Try giving your name in the field**

Step 3: Go to source code and analyse the vulnerability in the code



Source code:

localhost/DVWA/vulnerabilities/view_source.php?id=xss_r&security=low

Reflected XSS Source

vulnerabilities/xss_r/source/low.php

```php
<?php

header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}

?>
```

Code Review:

header("X-XSS-Protection: 0");

Disables browser's built-in XSS protection (like in Chrome).

$_GET['name']

Input is taken directly from the URL query parameter.

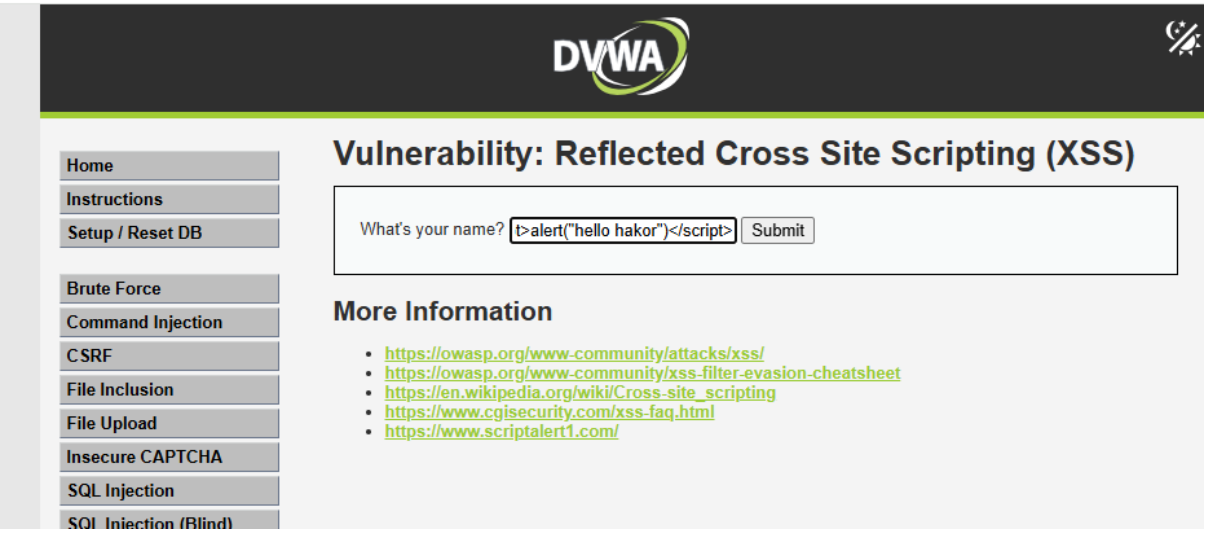
echo '<pre>Hello ' . $_GET['name'] . '</pre>';

User input is echoed without sanitization or encoding.

This creates a vulnerability because the browser will interpret any HTML/JavaScript in the input.

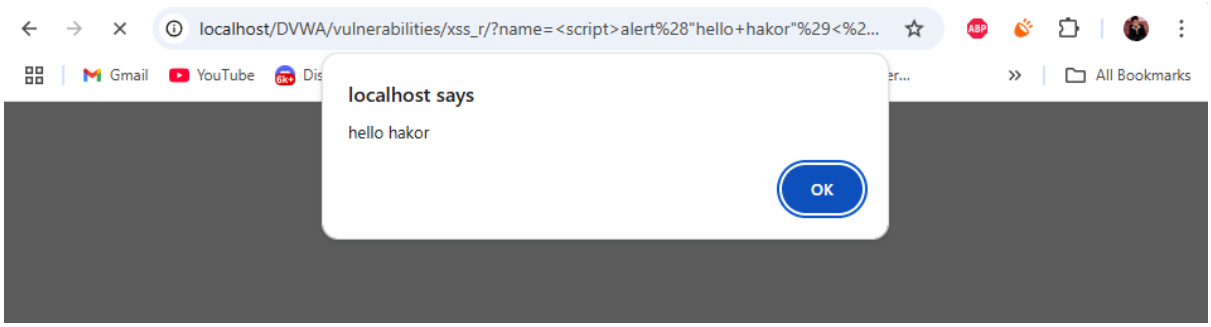
Step 4: Perform XSS(Reflected) attack

Payload: <script>alert("hello hakor")</script>



Step 5: Analyse the URL and popup

Result:


localhost/DVWA/vulnerabilities/xss_r/?name=<script>alert%28"hello+hakor"%29<%2...
Gmail
YouTube
All Bookmarks
localhost says
hello hakor
OK