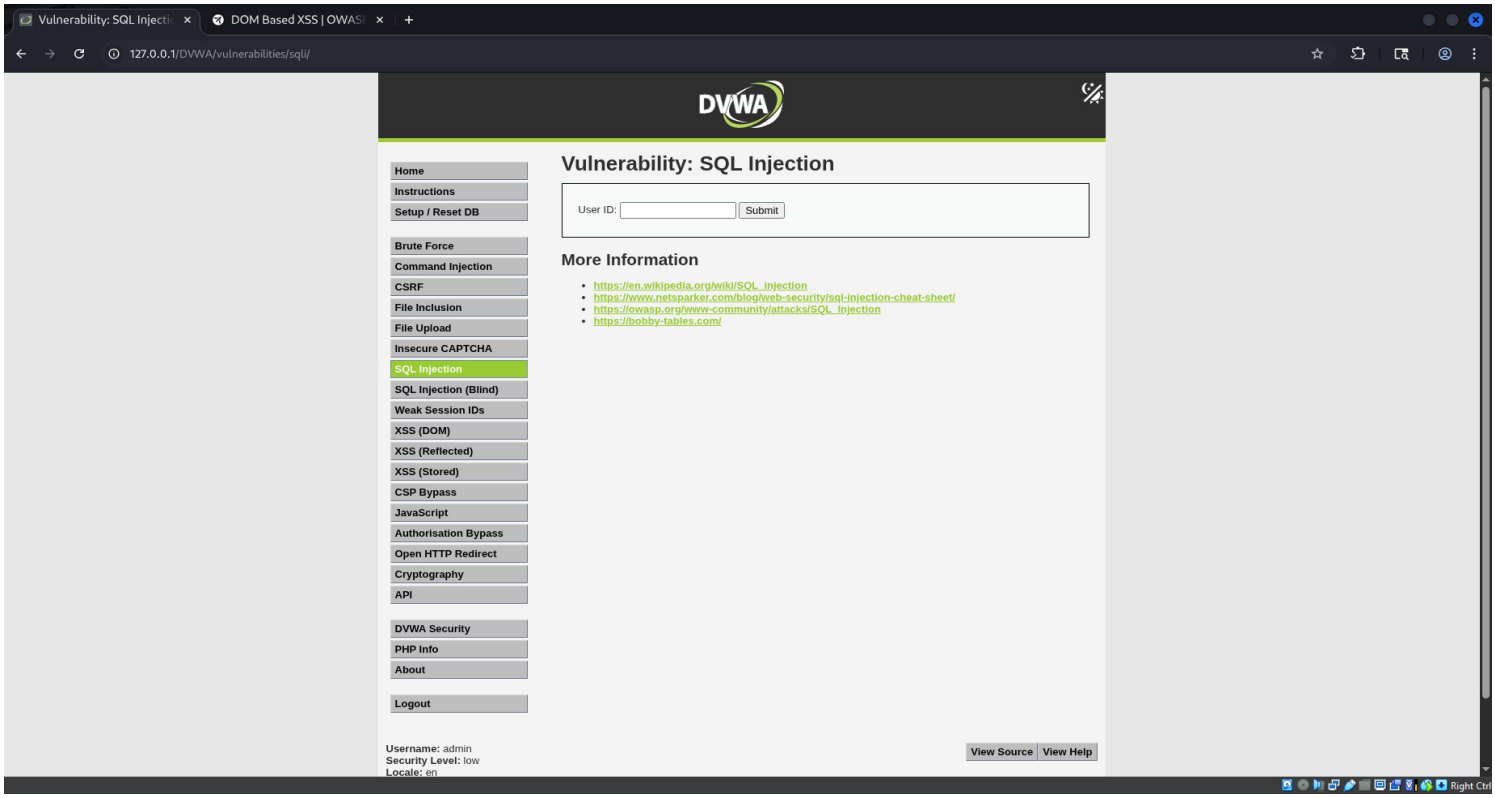
SQL INJECTION:



The above screenshot represent a DVWA*(Damn vulnerable web application),*the interface shows the input field where user enters some text.

Objective

There are 5 users in the database, with id's from 1 to 5. Your mission... to steal their passwords via SQLi.

This vulnerability retrives the data from database by processing the SQL statements from the user input field.This happens because the input the user entered is not validated(checked) whether the user entered correct input or not.

The reason for this failure lies in the source code provided below, where the code represent the id parameter which is not verified i.e, the boundaries for holding what type of data the id parameter hold is not defined by the developer.

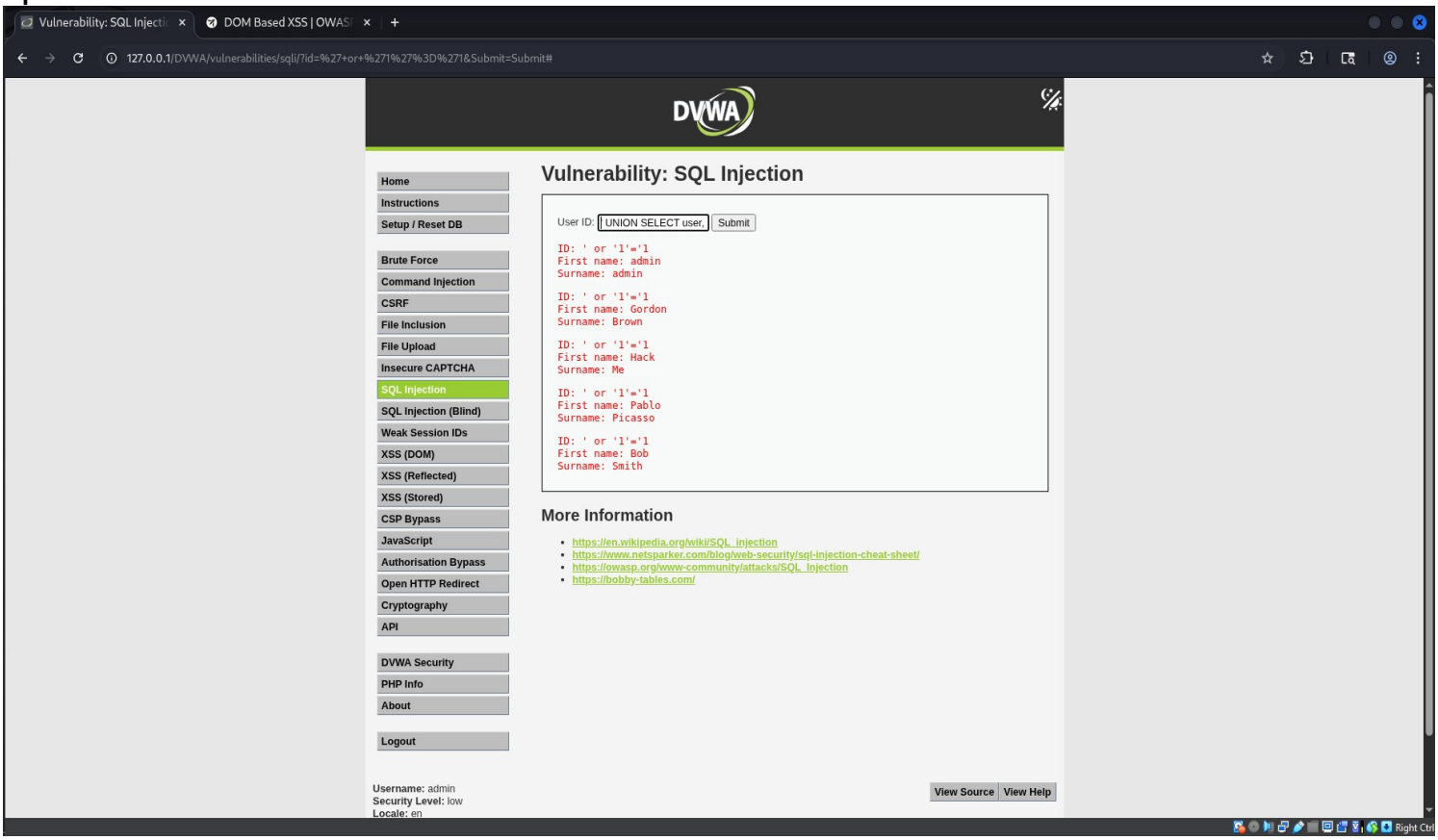
Thus this enable the malicious user to pass the SQL statements into the input field of web application such that the app process the input as an SQL statement and directly queries the database allowing the malicious user to bypass the database authentication for retrieving the data.

Damn Vulnerable Web Application (DVWA)Source :: Damn Vulnerable Web Application (DVWA) - Chromium

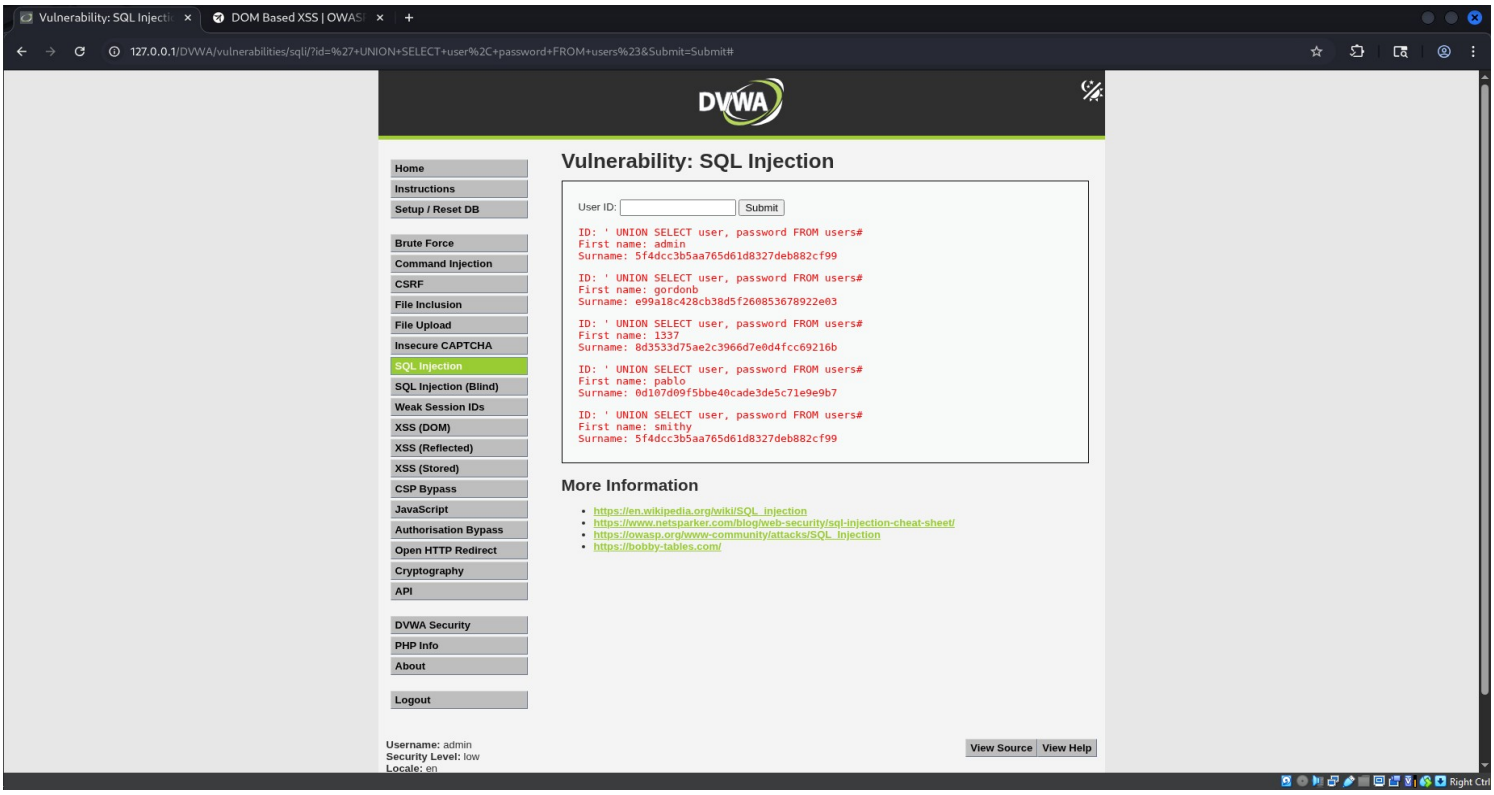
127.0.0.1/DVWA/vulnerabilities/view_source.php?id=sqli&security=low

```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["___

            // 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>";
                }
```

Screenshot showing the vulnerable code that do not validate the id parameter. In addition to that malicious user can also able to retrieve the passwords of the users present in the database.He/she may also know the table names/tables present in the database.



The above screenshot showing the Union attack performed via SQL injection. The statement ‘ UNION SELECT user , password FROM users# code tells the database to retrieve the usernames along with the passwords.But the passwords will be output as hashes as shown in below screenshot.

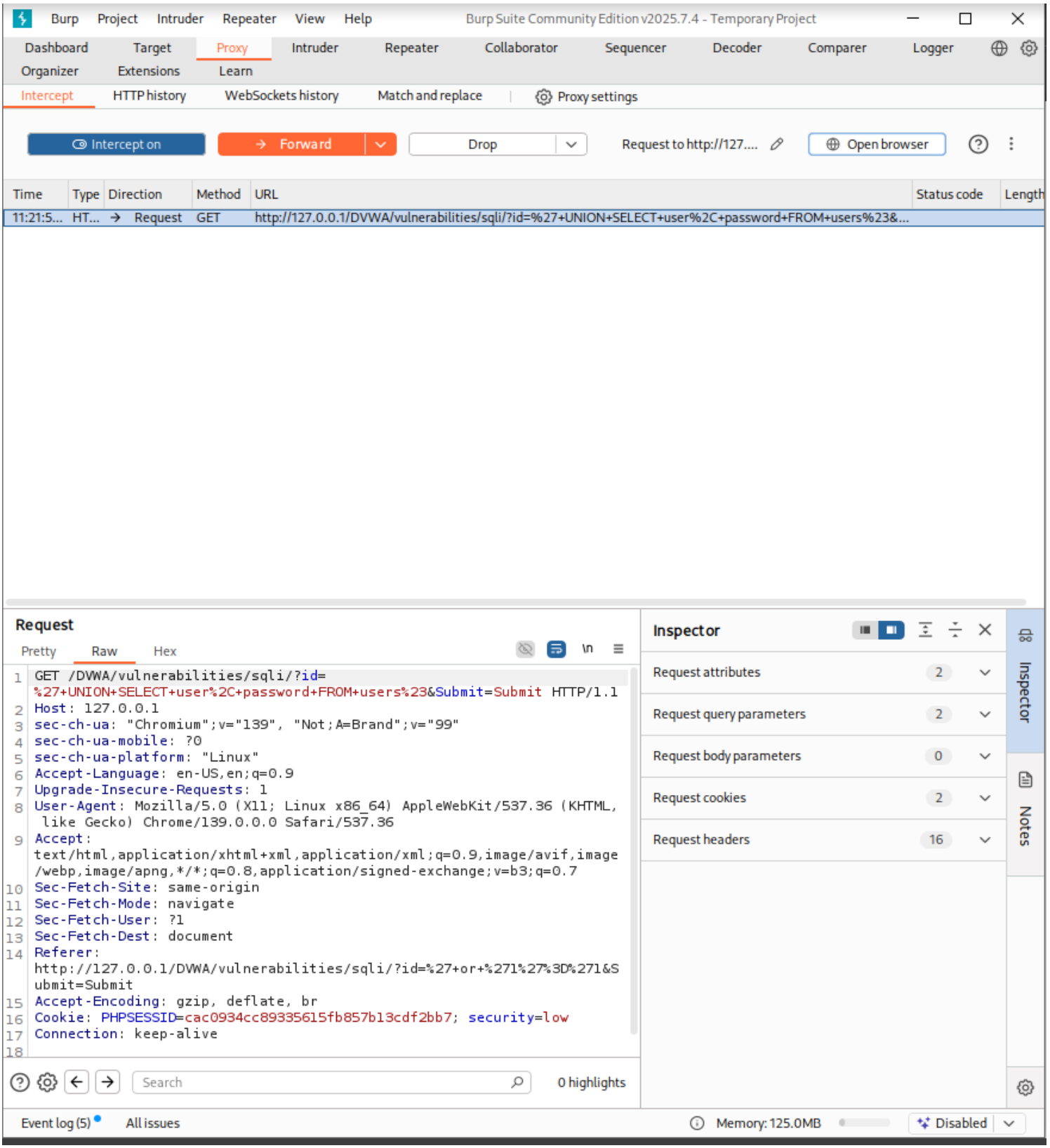


This UNION attack also uses the same id attibute.
SOLUTION: The input field of the user must be validated to a correct format so that input doesn’t process it to sql statements.

SQL Injection Source

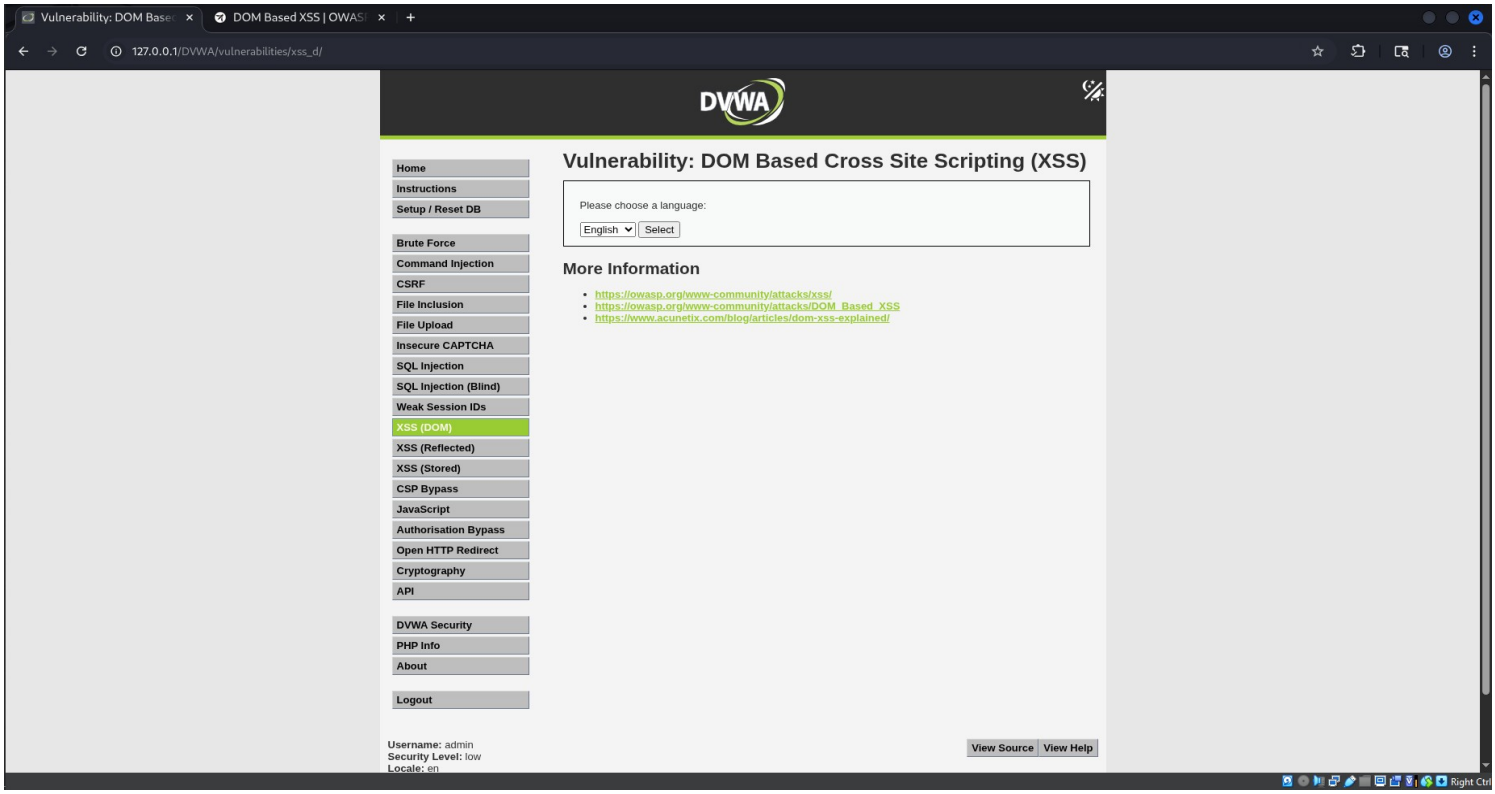
vulnerabilities/sqli/source/impossible.php

```
<?php

if( isset( $_GET[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $id = $_GET[ 'id' ];

    // Was a number entered?
    if(is_numeric( $id )) {
        $id = intval ($id);
        switch ($_DVWA['SQLI_DB']) {
            case MYSQL:
                // Check the database
                $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
                $data->bindParam( ':id', $id, PDO::PARAM_INT );
                $data->execute();
                $row = $data->fetch();

                // Make sure only 1 result is returned
                if( $data->rowCount() == 1 ) {
                    // 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>";
                }
                break;
            case SQLITE:
                global $sqlite_db_connection;

                $stmt = $sqlite_db_connection->prepare('SELECT first_name, last_name FROM users WHERE user_id = :id LIMIT 1;' );
                $stmt->bindValue(':id',$id,SQLITE3_INTEGER);
                $result = $stmt->execute();
                $result->finalize();
                if ($result !== false) {
                    // There is no way to get the number of rows returned
                    // This checks the number of columns (not rows) just
                    // as a precaution, but it won't stop someone dumping
                    // multiple rows and viewing them one at a time.

                    $num_columns = $result->numColumns();
                    if ($num_columns == 2) {
                        $row = $result->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>";
```

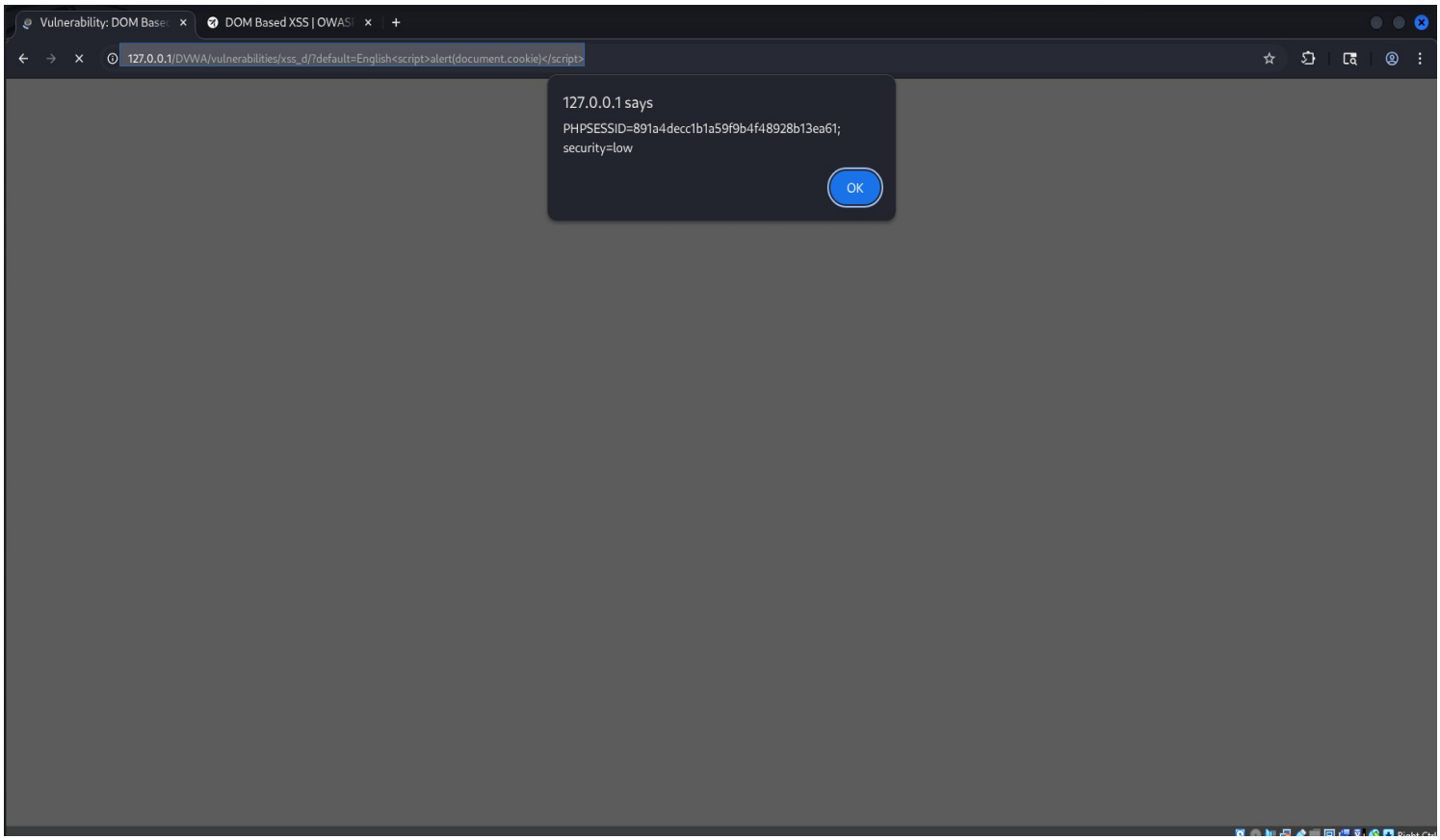


Burp suite result showing the HTTP GET request to the web application server containing the SQL Union code exploit.

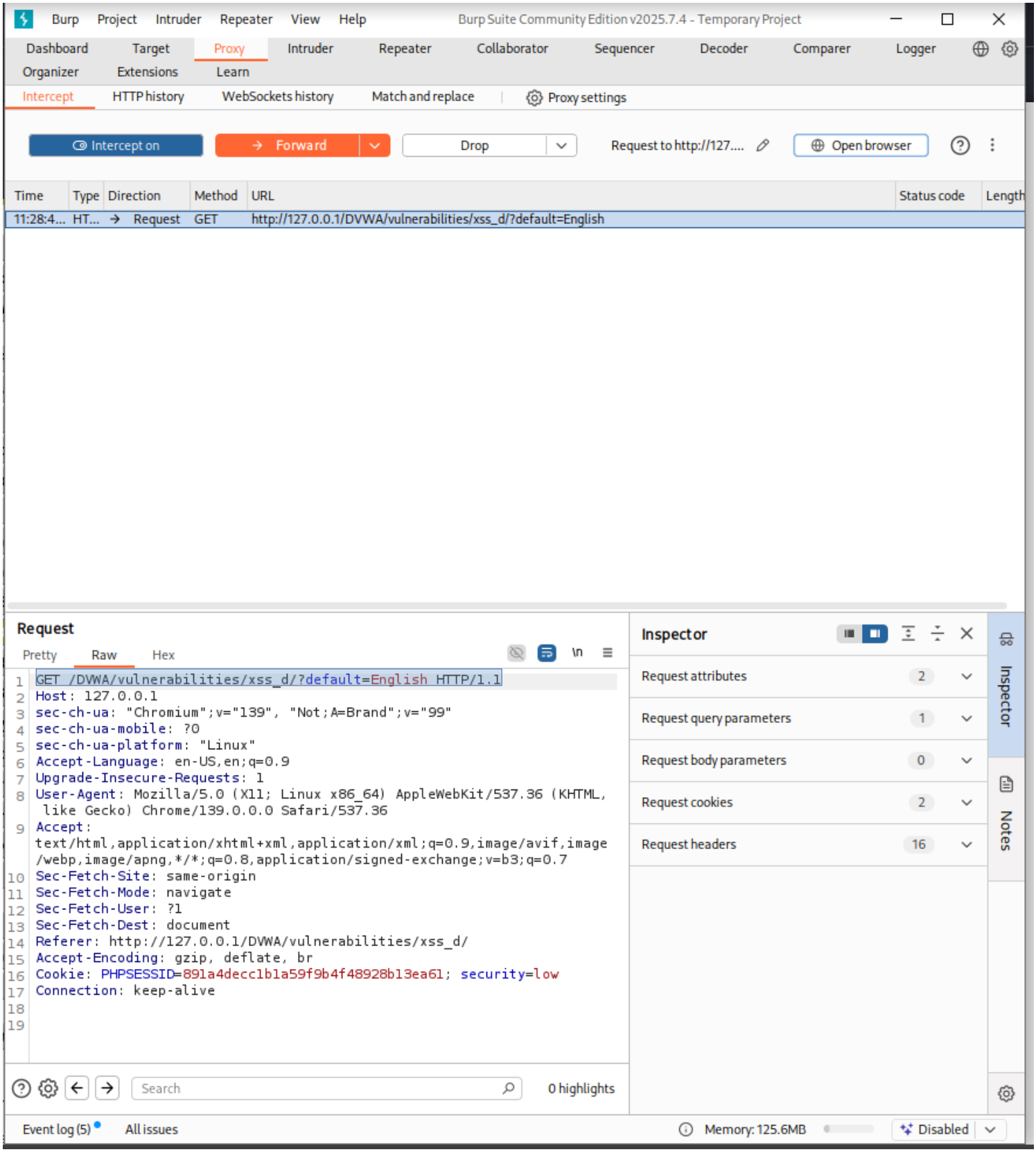
XSS[Cross-site Scripting]:



This is the DOM*[Document object model]* based attack,where user can manipulate the URL or the HTTP GET request to extract the session id/cookies by injecting the script from client side.



The above screenshot represent the exploit URL manipulation to extract the session id/cookies of users.



The above highlighted text is the field where the attacker modifies the request and Enters the malicious script instructions.

Solution:

> The contents taken from the URL are encoded by default by most browsers which prevents any injected JavaScript from being executed.

Top 10 Web Application Security Risks

There are three new categories, four categories with naming and scoping changes, and some consolidation in the Top 10 for 2021.