TRƯỜNG ĐẠI HỌC KHOA HỌC

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN MÔN HỌC

AN NINH MẠNG

SQL Injection (DVWA)

Giáo viên hướng dẫn:
Ths. Võ Việt Dũng

Sinh viên thực hiện :
1. Đoàn Quang Huy
2. Phan Duy Minh Thắng
3. Hoàng Ngọc Hưng
4. Ngô Anh Bảo
5. Hồ Đắc Trường An

Nhóm : 14
Lớp: 2
Năm học: 2025-2026

Huế, 01/2026

MỤC LỤC

MỞ ĐẦU

Lý do chọn đề tài

Trong bối cảnh chuyển đổi số diễn ra mạnh mẽ, các ứng dụng web đã và đang trở thành hạ tầng cốt lõi cho hầu hết các lĩnh vực như thương mại điện tử, ngân hàng, giáo dục, y tế và quản lý nhà nước. Cùng với sự phát triển nhanh chóng đó, vấn đề an toàn thông tin ngày càng trở nên cấp thiết khi số lượng các cuộc tấn công mạng gia tăng cả về quy mô lẫn mức độ tinh vi. Theo các báo cáo an ninh mạng toàn cầu và bảng xếp hạng OWASP Top 10, lỗ hổng SQL Injection luôn nằm trong nhóm các lỗ hổng nguy hiểm và phổ biến nhất, đe dọa trực tiếp đến tính bảo mật, toàn vẹn và sẵn sàng của hệ thống thông tin [1], [5], [6].

SQL Injection cho phép kẻ tấn công chèn các câu lệnh SQL độc hại vào truy vấn của ứng dụng, từ đó có thể đọc, chỉnh sửa, xóa dữ liệu trong cơ sở dữ liệu, thậm chí chiếm quyền điều khiển toàn bộ hệ thống. Nhiều sự cố rò rỉ dữ liệu lớn trên thế giới đã bắt nguồn từ việc khai thác lỗ hổng này, gây thiệt hại nghiêm trọng về kinh tế, uy tín và pháp lý cho các tổ chức, doanh nghiệp.

Nhận thức được mức độ nguy hiểm và tính thời sự của vấn đề, nhóm lựa chọn đề tài "SQL Injection (DVWA)" nhằm nghiên cứu có hệ thống về bản chất của lỗ hổng, phương thức khai thác cũng như các giải pháp phòng chống hiệu quả, qua đó góp phần nâng cao nhận thức và kỹ năng đảm bảo an toàn cho các hệ thống ứng dụng web.

Mục tiêu nghiên cứu

- Về lý thuyết: Làm rõ khái niệm, cơ chế hoạt động, phân loại các dạng tấn công SQL Injection phổ biến và phân tích mức độ nguy hiểm của từng dạng.
- Về thực nghiệm: Xây dựng môi trường giả lập thông qua DVWA (Damn Vulnerable Web Application), tiến hành mô phỏng các kịch bản tấn công tiêu biểu như Login Bypass, Error-based SQL Injection, từ đó minh họa trực quan quy trình khai thác lỗ hổng.

- Về giải pháp: Phân tích nguyên nhân tồn tại lỗ hổng trong mã nguồn, đề xuất các biện pháp phòng chống như Prepared Statement, Validation đầu vào, cơ chế phân quyền, mã hóa dữ liệu, góp phần nâng cao mức độ an toàn cho hệ thống.

Đối tượng và phạm vi nghiên cứu

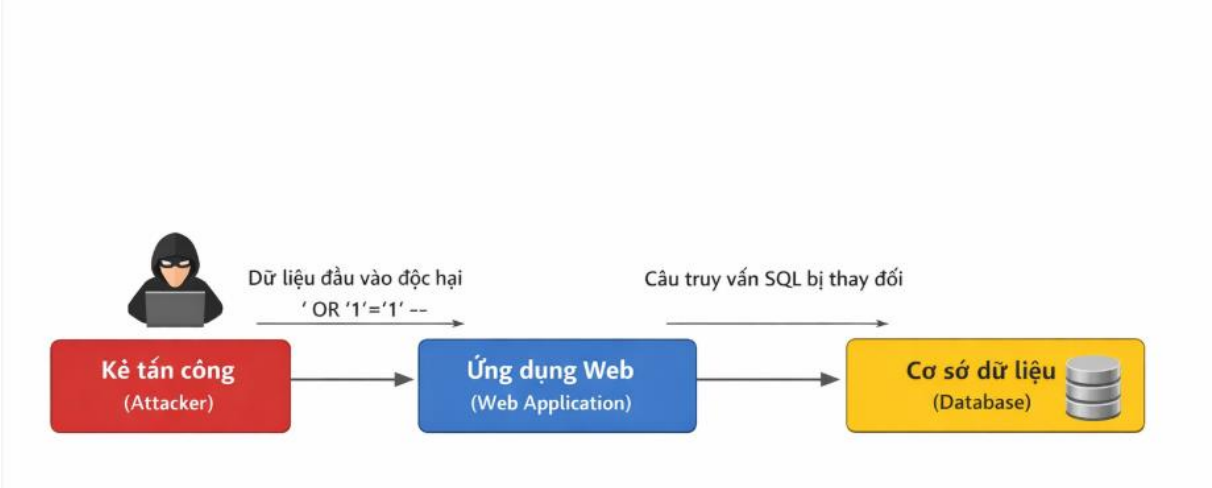
- Đối tượng nghiên cứu: Lỗ hổng bảo mật SQL Injection trong các ứng dụng web sử dụng ngôn ngữ lập trình PHP và hệ quản trị cơ sở dữ liệu MySQL.
- Phạm vi nghiên cứu: Nghiên cứu và thực nghiệm được tiến hành trên môi trường mô phỏng DVWA, tập trung vào việc khai thác và phòng chống các dạng SQL Injection phổ biến. Đề tài không đi sâu vào các kỹ thuật tấn công nâng cao trên hệ thống thực tế mà chủ yếu phục vụ mục đích học tập, nghiên cứu và nâng cao nhận thức về an toàn thông tin.

CHƯƠNG 1 - CƠ SỞ LÝ THUYẾT VỀ SQL INJECTION

1.1. Khái niệm SQL Injection

- SQL Injection là kỹ thuật tấn công cho phép kẻ tấn công chèn các câu lệnh SQL độc hại vào dữ liệu đầu vào của ứng dụng, từ đó làm thay đổi cấu trúc và logic của câu truy vấn SQL ban đầu. Khi ứng dụng không kiểm soát hoặc lọc dữ liệu đầu vào đúng cách, các tham số do người dùng cung cấp sẽ được ghép trực tiếp vào câu lệnh SQL và gửi tới hệ quản trị cơ sở dữ liệu để thực thi, tạo điều kiện cho kẻ tấn công vượt qua cơ chế xác thực, truy xuất trái phép dữ liệu, chỉnh sửa hoặc xóa thông tin quan trọng trong hệ thống [4], [6].

- Trên thực tế, SQL Injection không chỉ đe dọa tính bảo mật của hệ thống mà còn ảnh hưởng nghiêm trọng đến tính toàn vẹn và tính sẵn sàng của dữ liệu. Một cuộc tấn công SQL Injection thành công có thể dẫn đến rò rỉ thông tin cá nhân, thông tin tài chính, dữ liệu doanh nghiệp, thậm chí cho phép kẻ tấn công chiếm quyền điều khiển máy chủ ứng dụng. Theo các báo cáo an ninh mạng, SQL Injection luôn nằm trong nhóm những lỗ hổng phổ biến và nguy hiểm nhất đối với các ứng dụng web trong nhiều năm liên tiếp [5], [6].



Hình 1.1. Biểu đồ mô tả SQL Injection

1.2. Cơ chế hoạt động của SQL Injection

- Cơ chế của SQL Injection xuất phát từ việc ứng dụng xây dựng câu lệnh SQL bằng cách ghép trực tiếp dữ liệu đầu vào của người dùng mà không thực hiện các bước

kiểm tra, lọc và bảo vệ cần thiết. Khi dữ liệu không được xử lý an toàn, kẻ tấn công có thể chèn các đoạn mã SQL độc hại vào tham số đầu vào nhằm làm thay đổi cấu trúc và logic của câu truy vấn ban đầu [4], [5]:

- Thông thường, trong các ứng dụng web, dữ liệu người dùng (username, password, id, search, v.v.) được truyền đến máy chủ và gắn vào câu lệnh SQL để truy vấn cơ sở dữ liệu. Khi không có cơ chế bảo vệ thích hợp, dữ liệu này không còn là giá trị đầu vào thuần túy mà trở thành một phần của câu lệnh điều khiển, khiến hệ quản trị cơ sở dữ liệu không thể phân biệt đâu là dữ liệu hợp lệ, đâu là lệnh tấn công [4].

SELECT * FROM users WHERE username = '$username' AND password = '$password';

- Nếu kẻ tấn công nhập vào trường username đoạn mã: ' OR '1'='1 --, câu lệnh SQL thực thi trong hệ thống sẽ trở thành:

SELECT * FROM users WHERE username = '' OR '1'='1' --' AND password = '';

Dấu -- trong SQL được sử dụng để bình luận (comment) phần còn lại của câu lệnh, khiến điều kiện kiểm tra mật khẩu bị vô hiệu hóa. Do biểu thức '1'='1' luôn đúng, hệ thống sẽ trả về bản ghi đầu tiên trong bảng người dùng, cho phép đăng nhập trái phép mà không cần mật khẩu hợp lệ [4], [5].

Cơ chế này cho thấy SQL Injection không đơn thuần là lỗi cú pháp, mà là lỗi thiết kế bảo mật trong quá trình phát triển ứng dụng web.

1.3. Các dạng lỗi phổ biến trong SQL Injection

Dựa trên kỹ thuật khai thác và phản hồi của hệ thống, SQL Injection thường được chia thành các dạng sau [7]:

1.3.1. SQL Injection bypass đăng nhập (Authentication Bypass)

Đây là dạng tấn công phổ biến nhất, thường xảy ra tại các form đăng nhập. Kẻ tấn công chèn điều kiện logic luôn đúng để vượt qua bước xác thực người dùng.

Ví dụ payload: ' OR '1'='1 –

Hậu quả:

- Đăng nhập với quyền người dùng bất kỳ
- Truy cập các chức năng nội bộ
- Làm bàn đạp cho các tấn công nâng cao hơn

1.3.2. Error-based SQL Injection

- Dạng tấn công này dựa vào thông báo lỗi SQL mà hệ thống trả về. Thông qua lỗi, kẻ tấn công có thể thu thập thông tin về cấu trúc cơ sở dữ liệu như tên bảng, tên cột hoặc loại DBMS đang sử dụng [3], [6].

Thông tin có thể thu thập được:

- Tên bảng
- Tên cột
- Kiểu dữ liệu
- Phiên bản DBMS

Đặc điểm:

- Dễ khai thác
- Phụ thuộc vào cấu hình hiển thị lỗi
- Ít hiệu quả nếu hệ thống đã tắt debug

1.3.3. Union-based SQL Injection

-Union-based SQL Injection sử dụng toán tử UNION để kết hợp kết quả của truy vấn gốc với truy vấn do kẻ tấn công tạo ra. Qua đó, dữ liệu từ các bảng khác có thể được hiển thị trực tiếp trên giao diện ứng dụng [5].

Ví dụ payload: ' UNION SELECT username, password FROM users –

Điều kiện khai thác:

- Số lượng cột phải khớp
- Kiểu dữ liệu tương thích

1.3.4. Boolean-based Blind SQL Injection

Trong trường hợp ứng dụng không hiển thị lỗi và không trả dữ liệu trực tiếp, kẻ tấn công sẽ dựa vào sự khác biệt trong phản hồi đúng/sai của ứng dụng để suy đoán thông tin.

Ví dụ:

Kiểm tra đúng: ' AND 1=1 --

Kiểm tra sai: ' AND 1=2 --

1.3.5. Time-based Blind SQL Injection

Dạng này dựa vào thời gian phản hồi của hệ thống. Khi điều kiện đúng, hệ thống sẽ bị trì hoãn trong một khoảng thời gian nhất định [7].

Ví dụ payload: ' AND SLEEP(5) --

Nếu trang web phản hồi chậm hơn bình thường, điều đó cho thấy câu điều kiện là đúng.

1.4. Các nguyên nhân gây ra lỗ hổng

Các nguyên nhân chính dẫn đến SQL Injection bao gồm:

- Không kiểm tra, xác thực và lọc dữ liệu đầu vào
- Xây dựng câu truy vấn bằng cách ghép chuỗi trực tiếp
- Không sử dụng Prepared Statement hoặc truy vấn tham số hóa
- Cấu hình hiển thị lỗi SQL không an toàn
- Thiếu kiến thức và quy trình lập trình an toàn (Secure Coding)
- Áp lực phát triển nhanh khiến bảo mật bị xem nhẹ trong giai đoạn đầu của dự án [4], [6]

CHƯƠNG 2 – THỰC HIỆN LOGIN BYPASS ĐƠN GIẢN TRÊN MÔI TRƯỜNG DVWA

2.1. Môi trường thực hiện

Trong mục khai thác DVWA ta chỉnh mức độ bảo mật của trang web xuống mức thấp nhất là security level = low và PHPIDS (PHP-Intrusion Detection System) = disabled để làm bài phân tích lỗ hổng và mô phỏng.

2.2. Tổng quan và mục tiêu của Login bypass

Tổng quan Login Bypass (hay còn gọi là vượt qua xác thực) là một lỗ hổng bảo mật nghiêm trọng phát sinh, xảy ra tại chức năng đăng nhập của ứng dụng có sử dụng SQL để chuyển tiếp đến cơ sở dữ liệu. Lỗi này thường xuất phát từ việc hệ thống không kiểm soát chặt chẽ dữ liệu đầu vào của câu truy vấn và quản lý lỏng lẻo các câu lệnh truy vấn khi tương tác với cơ sở dữ liệu [5].

Lợi dụng điều này bất kỳ mã code nào được nhập vào từ nguồn không đáng tin cậy vẫn được thực thi trên cơ sở dữ liệu và luôn trả về kết quả là đúng và hệ thống sẽ cho đăng nhập vào phiên rồi kết nối tới hệ thống. Cuối cùng login bypass cơ bản đã hoàn thành [7].

Mục tiêu tấn công Mục tiêu chính của kẻ tấn công khi khai thác lỗ hổng Login Bypass bao gồm:

1. Truy cập trái phép: Đăng nhập thành công vào hệ thống dưới danh nghĩa của người dùng hợp lệ hoặc quản trị viên (admin) mà không cần biết mật khẩu.
2. Leo thang đặc quyền: Từ việc chiếm được tài khoản quản trị, kẻ tấn công có thể toàn quyền kiểm soát ứng dụng.
3. Đánh cắp và thao túng dữ liệu: Sau khi thâm nhập, kẻ tấn công có thể xem, sửa, xóa dữ liệu nhạy cảm trong hệ thống.

2.3. Khai thác lỗ hổng của kỹ thuật

Bước 1: Quan sát giao diện Tại giao diện đăng nhập của DVWA, hệ thống yêu cầu người dùng nhập *Username* và *Password*.

Hình 2.1. Giao diện chức năng đăng nhập trên DVWA.

Bước 2: Phân tích mã nguồn. Dựa vào mã nguồn câu lệnh truy vấn xác thực người dùng có dạng như sau.

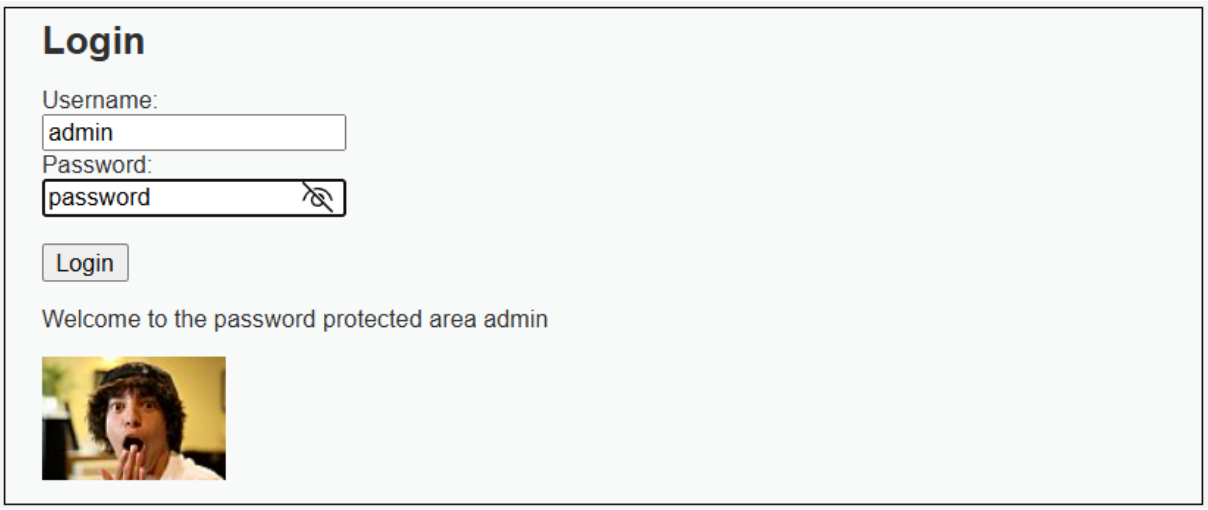
```php
<?php

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';";
```

Hình 2.2. Đoạn mã xử lý đăng nhập có lỗ hổng (Nguồn: DVWA)

Từ hình 2.2. qua đó câu truy vấn chỉ kiểm tra Username, Password có trùng với thông tin user, password được nhập từ form đăng nhập hay không. Từ đó có thể đưa ra phương án đăng nhập thành công mà không cần biết password là gì.

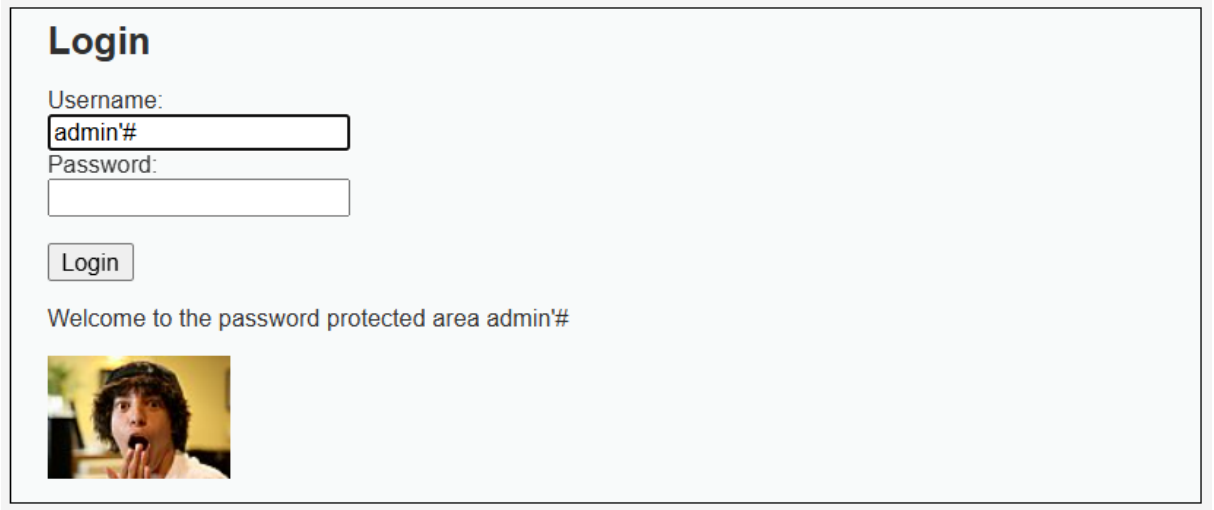
Bước 3: Thực hiện tấn công

Kịch bản 1: Nhập Username = admin và Password = password. Hệ thống hoạt động bình thường.



Hình 2.3. Đăng nhập thành công với mật khẩu hợp lệ.

Kịch bản 2: Giả sử kẻ tấn công biết tên đăng nhập là admin nhưng không biết mật khẩu. Kẻ tấn công nhập vào trường Username payload sau: admin' #.



Hình 2.4. Đăng nhập thành công không cần mật khẩu.

2.4. Giải thích cơ chế hoạt động

Từ đoạn mã nguồn trong Hình 2.2 có thể thấy, hệ thống xác thực người dùng của DVWA được xây dựng theo cách rất đơn giản, chỉ kiểm tra sự trùng khớp giữa dữ liệu người dùng nhập vào và dữ liệu lưu trong cơ sở dữ liệu mà không áp dụng bất kỳ cơ chế bảo vệ nào như lọc dữ liệu, mã hóa tham số hay truy vấn tham số hóa. Câu truy vấn gốc có dạng:

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

Khi ứng dụng ghép trực tiếp biến $user và $pass vào câu truy vấn, toàn bộ dữ liệu do người dùng nhập vào đều trở thành một phần của câu lệnh SQL được thực thi trên máy chủ cơ sở dữ liệu. Đây chính là nguyên nhân cốt lõi tạo ra lỗ hổng SQL Injection [4], [5], [6].

Trong kịch bản tấn công, kẻ tấn công nhập vào trường Username giá trị: admin' #

Khi đó câu truy vấn SQL thực tế được xây dựng trên hệ thống sẽ trở thành:

SELECT * FROM `users` WHERE user = 'admin'#' AND password = '...';

Trong hệ quản trị cơ sở dữ liệu MySQL, ký tự # được sử dụng để đánh dấu chú thích (comment)**.** Mọi nội dung nằm sau ký tự này sẽ bị bỏ qua và không được thực thi. Do đó, phần kiểm tra mật khẩu: AND password = '...' bị vô hiệu hóa hoàn toàn. Cơ sở dữ liệu lúc này chỉ còn thực hiện câu truy vấn:

SELECT * FROM `users` WHERE user = 'admin';

Nếu tài khoản admin tồn tại trong hệ thống, truy vấn trên sẽ trả về kết quả hợp lệ, khiến ứng dụng tin rằng quá trình xác thực thành công, mặc dù kẻ tấn công không hề biết mật khẩu thật. Như vậy, chỉ với một ký tự đặc biệt, kẻ tấn công đã có thể thay đổi hoàn toàn logic xác thực của hệ thống và đăng nhập trái phép [5], [7].

Phân tích sâu hơn về mức độ nguy hiểm

Cơ chế này cho thấy hệ thống đã không phân tách rõ ràng giữa dữ liệu và câu lệnh điều khiển, dẫn đến việc dữ liệu đầu vào của người dùng có thể can thiệp trực tiếp vào quá trình xử lý truy vấn của cơ sở dữ liệu. Đây là sai lầm nghiêm trọng trong thiết kế ứng dụng web, vi phạm các nguyên tắc cơ bản của lập trình an toàn [4], [6].

Nếu kẻ tấn công đăng nhập được vào tài khoản quản trị, toàn bộ hệ thống có thể bị chiếm quyền kiểm soát, bao gồm:

- Thay đổi hoặc xóa dữ liệu quan trọng
- Cài mã độc, tạo tài khoản ẩn
- Mở rộng tấn công sang các hệ thống khác trong mạng nội bộ

Điều này chứng minh rằng một lỗ hổng nhỏ trong chức năng đăng nhập có thể dẫn đến sự sụp đổ hoàn toàn của hệ thống bảo mật.

CHƯƠNG 3 – KHAI THÁC UNION-BASED

3.1. Tổng quan về kỹ thuật khai thác Union-based

3.1.1. Khái niệm

Union-based SQL Injection là một kỹ thuật khai thác lỗ hổng bảo mật trong đó kẻ tấn công lợi dụng toán tử UNION (hoặc UNION ALL) của ngôn ngữ SQL để kết hợp kết quả của câu truy vấn hợp lệ do ứng dụng tạo ra với kết quả của một câu truy vấn do chính kẻ tấn công xây dựng. Thông qua cơ chế này, dữ liệu ngoài phạm vi cho phép của ứng dụng – đặc biệt là các thông tin nhạy cảm như tài khoản người dùng, mật khẩu, quyền truy cập và dữ liệu quản trị – có thể bị ép hiển thị trực tiếp trên giao diện web [5].

Mục đích chính của kỹ thuật Union-based là vượt qua các giới hạn truy xuất dữ liệu mà ứng dụng đặt ra, buộc hệ thống trả về thông tin từ các bảng khác trong cơ sở dữ liệu (chẳng hạn bảng users, passwords, admin, …) thay vì chỉ hiển thị dữ liệu hợp lệ theo chức năng ban đầu. Điều này khiến kẻ tấn công có thể thu thập được cấu trúc cơ sở dữ liệu, danh sách người dùng, thông tin đăng nhập và nhiều dữ liệu nhạy cảm khác chỉ với một số lượng rất ít truy vấn, gây ra nguy cơ rò rỉ dữ liệu nghiêm trọng và làm suy giảm toàn bộ hệ thống bảo mật của ứng dụng [5], [7].

3.1.2. Điều kiện khai thác

Theo nguyên lý hoạt động của hệ quản trị cơ sở dữ liệu SQL, để một cuộc tấn công Union-based SQL Injection có thể thực hiện thành công, câu truy vấn do kẻ tấn công tạo ra bắt buộc phải thỏa mãn hai điều kiện nghiêm ngặt sau đây [5], [7]:

- Thứ nhất, số lượng cột phải tương đồng**.** Câu lệnh SELECT của kẻ tấn công phải trả về đúng số lượng cột như câu truy vấn gốc của ứng dụng. Nếu số lượng cột không khớp, hệ quản trị cơ sở dữ liệu sẽ phát sinh lỗi và từ chối thực thi câu lệnh. Do đó, trong quá trình khai thác, kẻ tấn công thường phải thử nghiệm nhiều lần bằng các truy vấn giả để xác định chính xác số lượng cột của truy vấn ban đầu.

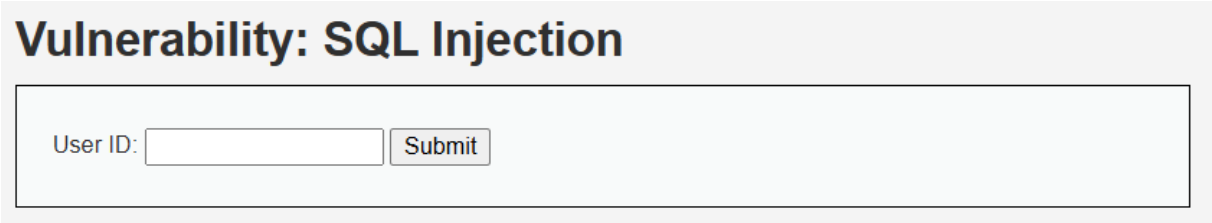
- Thứ hai, kiểu dữ liệu phải tương thích. Các cột tại cùng một vị trí trong hai câu lệnh SELECT phải có kiểu dữ liệu giống nhau hoặc có khả năng chuyển đổi tương thích (ví dụ: chuỗi, số, NULL). Nếu kiểu dữ liệu không tương thích, cơ sở dữ liệu sẽ không

thể kết hợp hai tập kết quả và quá trình tấn công sẽ thất bại. Vì vậy, kẻ tấn công thường sử dụng các giá trị như NULL, chuỗi ký tự hoặc các hàm chuyển đổi kiểu để đảm bảo sự tương thích dữ liệu trong từng vị trí cột.

Hai điều kiện này quyết định trực tiếp đến khả năng thành công của cuộc tấn công Union-based và cũng là cơ sở để kẻ tấn công xây dựng chiến lược dò tìm cấu trúc cơ sở dữ liệu một cách có hệ thống và hiệu quả [5], [7].

3.2. Quy trình khai thác thực tế

Quan sát giao diện khai thác SQL Injection cho thấy người dùng cần nhập ID để hệ thống hiện thông tin của ID vừa nhập



Hình 3.1. Giao diện khai thác.

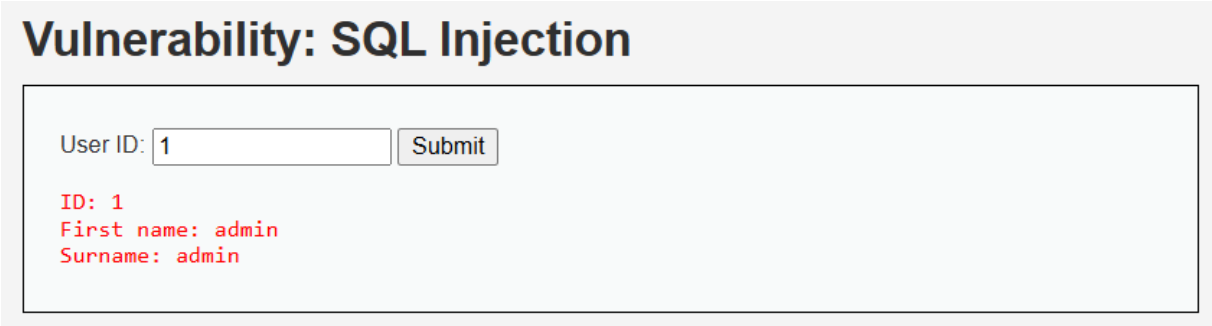
Phân tích mã nguồn. Dựa vào mã nguồn câu lệnh truy vấn ID có dạng như sau.

```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["___

    // Get results
    while( $row = mysqli_fetch_assoc( $result ) ) {
```

Hình 3.2. Đoạn mã xử lý ID có lỗ hổng.

Khai thác:

Bước 1: Thử nhập một vài số nguyên để xem kết quả trả về là gì



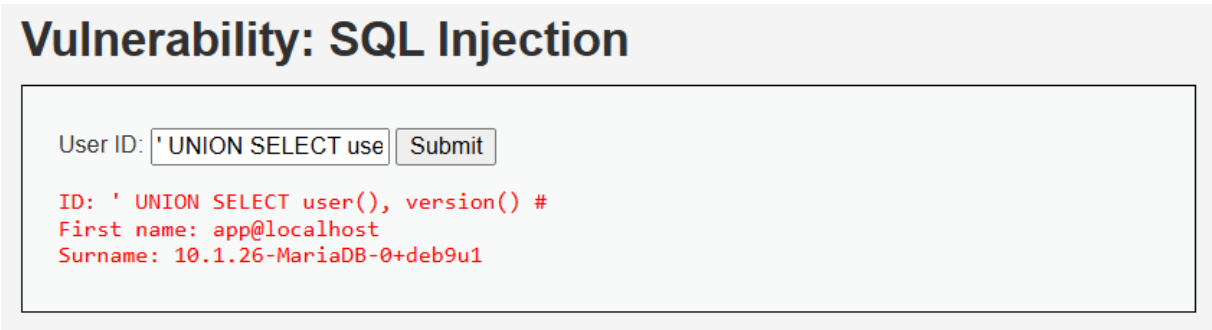
Hình 3.3. Thông tin trả về khi id = 1.

Trong bảng users mà câu truy vấn sử dụng trả về hai cột đó là First name và Surname qua đó thấy trong bảng user sẽ có chứa các dữ liệu quan trong về user dữ liệu này rất quan trọng trong hệ thống và khi người tấn công biết tất cả dữ liệu trong bảng này thì sẽ ảnh hưởng rất lơn tới an ninh hệ thống và người dùng hệ thống.

Bước 2: Thu thập thông tin định danh hệ thống

Sau khi xác định được các vị trí hiển thị dữ liệu trên giao diện web (First name và Surname), kẻ tấn công tiến hành thay thế các con số giả định bằng các hàm đặc biệt để thu thập thông tin về môi trường máy chủ.

Payload: ' UNION SELECT user(), version() #

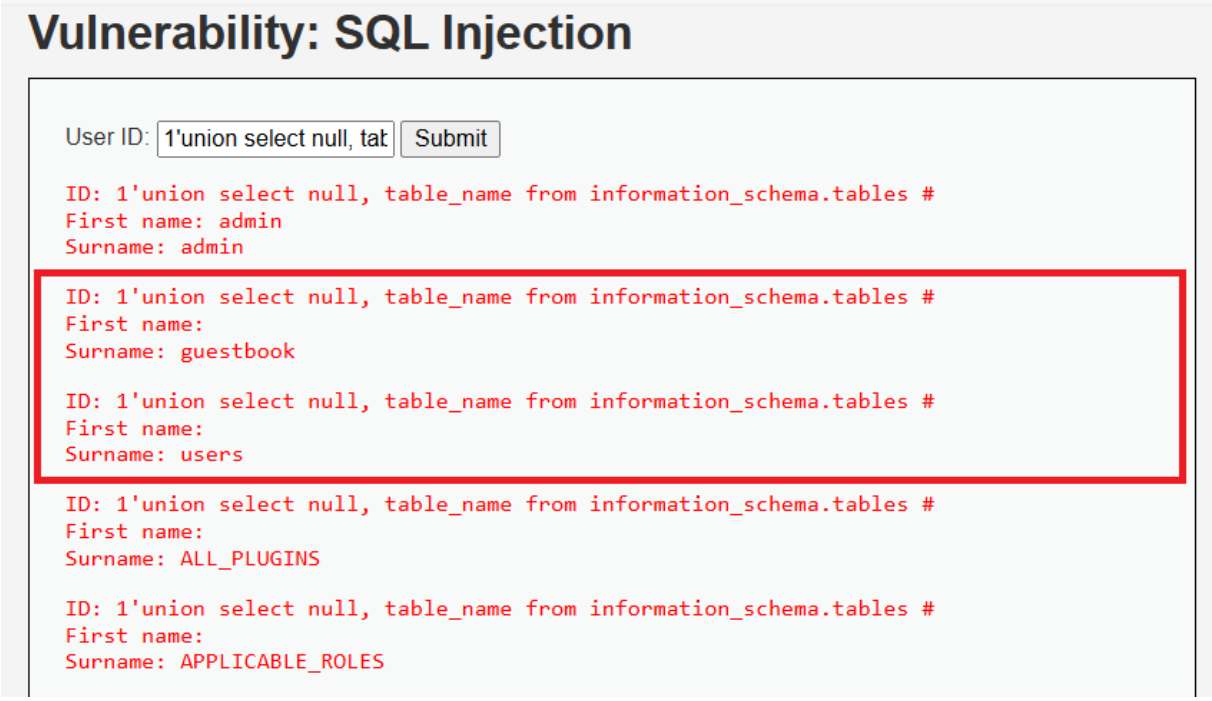


Hình 3.4. Thông tin của hệ thống.

Đánh giá rủi ro: Việc lộ lọt phiên bản hệ quản trị (Version Disclosure) giúp kẻ tấn công dễ dàng tra cứu các lỗ hổng bảo mật đã được công bố (CVE) tương ứng với phiên bản MariaDB 10.1.26 để lên kế hoạch tấn công sâu hơn [7].

Bước 3: Tìm các bảng khác trong cơ sở dữ liệu

Payload: 1'union select null, table_name from information_schema.tables #



Hình 3.5. Bảng có trong cơ sở dữ liệu.

Trong cơ sở dữ liệu của hệ thống có hai bảng chính là users và guestbook qua đó thấy được phần trọng tâm khai thác sẽ ở trong bảng users. Tiếp theo sẽ liệt kê tất cả các cột tồn tại trong bảng users.

Bước 4: Liệt kê danh sách các cột trong bảng users

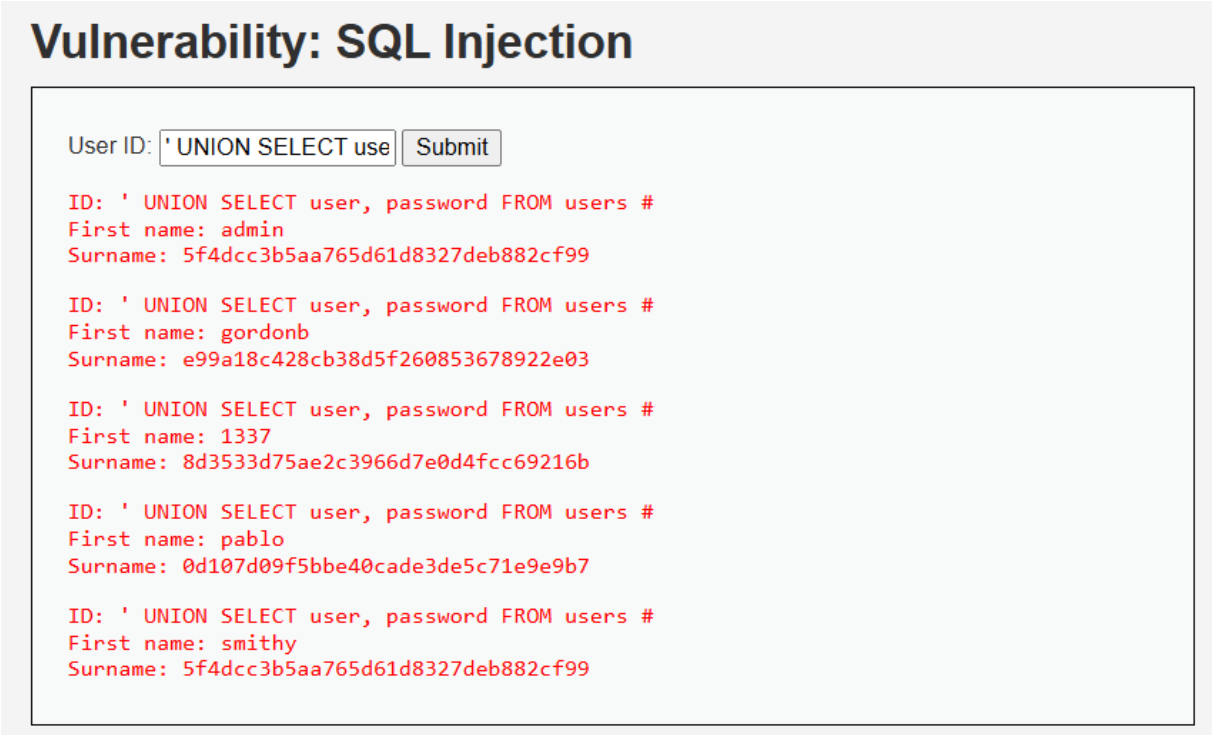
Payload: ' UNION SELECT 1, group_concat(column_name) FROM information_schema.columns WHERE table_name = 'users' #

Hình 3.6. Các cột trong bảng users.

Trong kết quả trả về có 2 cột là user và password là 2 cột chứa thông tin đăng nhập của người dùng đây là phần quan trọng nhất trong cuộc tấn công dữ liệu và có thể truy cập với tư cách là người dùng.

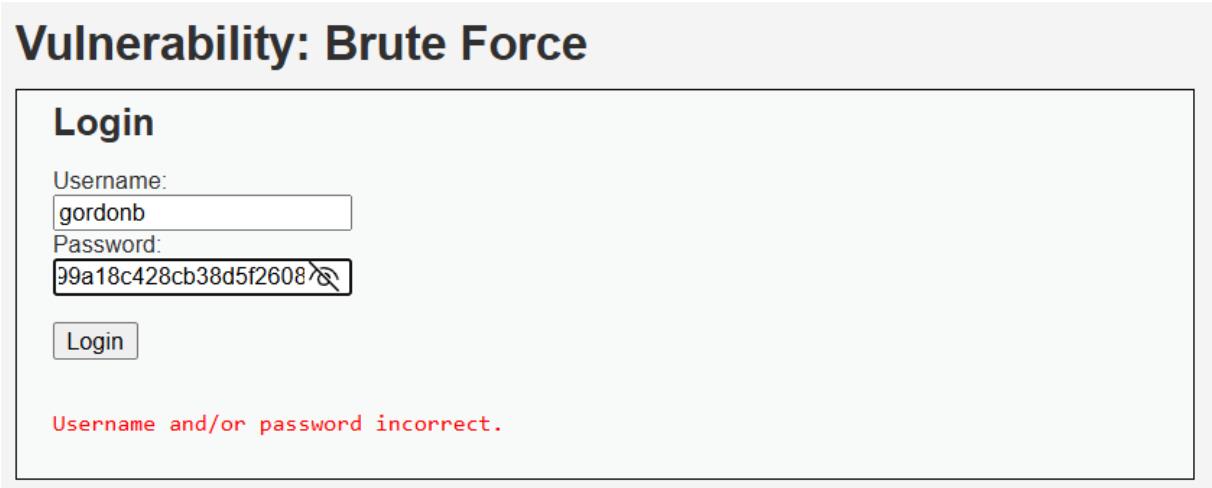
Bước 5: Trích xuất dữ liệu nhạy cảm. Sau khi đã có đầy đủ thông tin (Tên bảng: users, Tên cột: user, password), ta thực hiện cuộc tấn công cuối cùng để lấy toàn bộ dữ liệu.

Payload: ' UNION SELECT user, password FROM users #



Hình 3.7. Danh sách tài khoản.

Từ danh sách tài khoản đã có đăng nhập thử với user = gordonb và password = e99a18c428cb38d5f260853678922e03

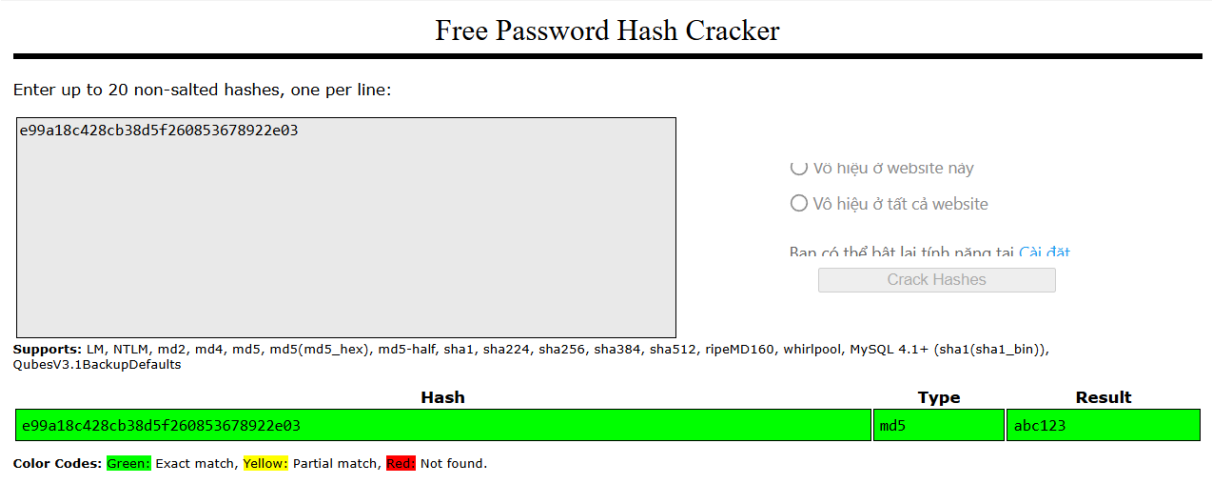


Hình 3.8. Đăng nhập user = gordonb.

Dữ liệu mật khẩu thu được (e99a18c428cb38d5f260853678922e03) không phải là văn bản gốc mà là chuỗi mã hóa băm (Hash). Dựa trên độ dài 32 ký tự và chỉ gồm số cùng các chữ cái a-f, ta xác định đây là thuật toán MD5.

Bước 6: Giải mã mật khẩu

Để giải mã mật khẩu trên, sử dụng công cụ giải mã trực tuyến (https://crackstation.net/) để bẻ khóa được mật khẩu (e99a18c428cb38d5f260853678922e03).



Hình 3.9. Bẻ khóa mật khẩu bằng thuật toán MD5.

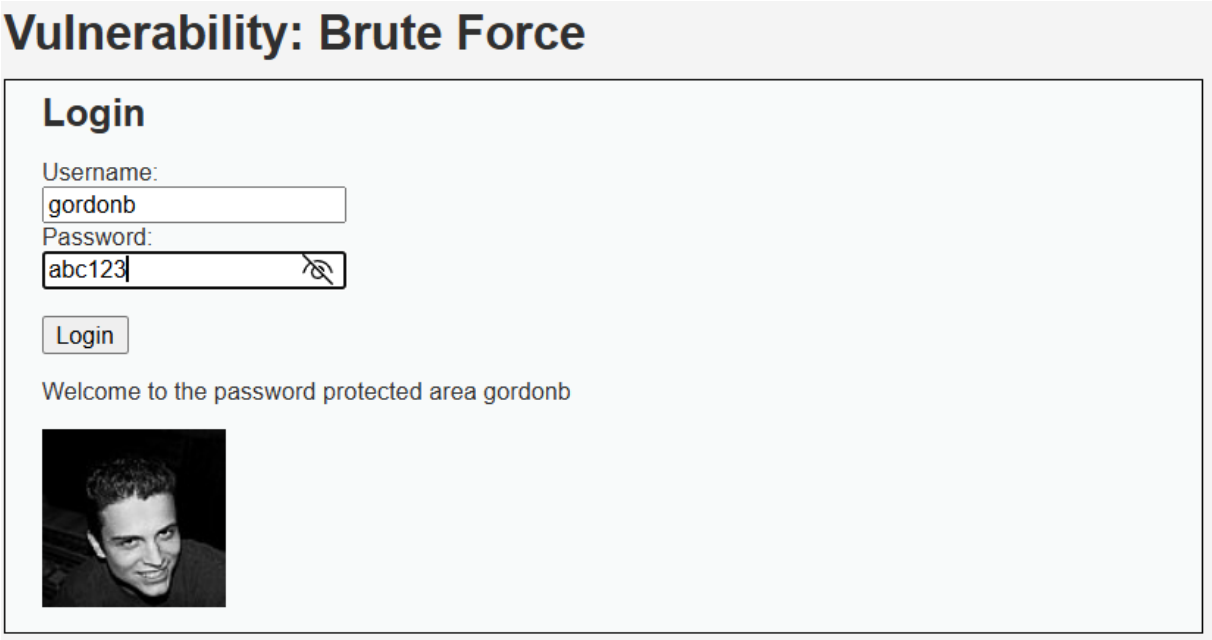
Danh sách đăng nhập hoàn chỉnh sau khi đã được giải mã.

User	MD5	Password
admin	5f4dcc3b5aa765d61d8327deb882cf99	password
gordonb	e99a18c428cb38d5f260853678922e03	abc123
1337	8d3533d75ae2c3966d7e0d4fcc69216b	charley
pablo	0d107d09f5bbe40cade3de5c71e9e9b7	letmein
smithy	5f4dcc3b5aa765d61d8327deb882cf99	password

Bước 7: Tiến hành đăng nhập vào hệ thống bằng các user vừa khai thác được

Kịch bản 1: Đăng nhập với user và password từ bảng users. Đăng nhập lại với tư cách người dùng gordonb sau khi đã giải mã được mật khẩu

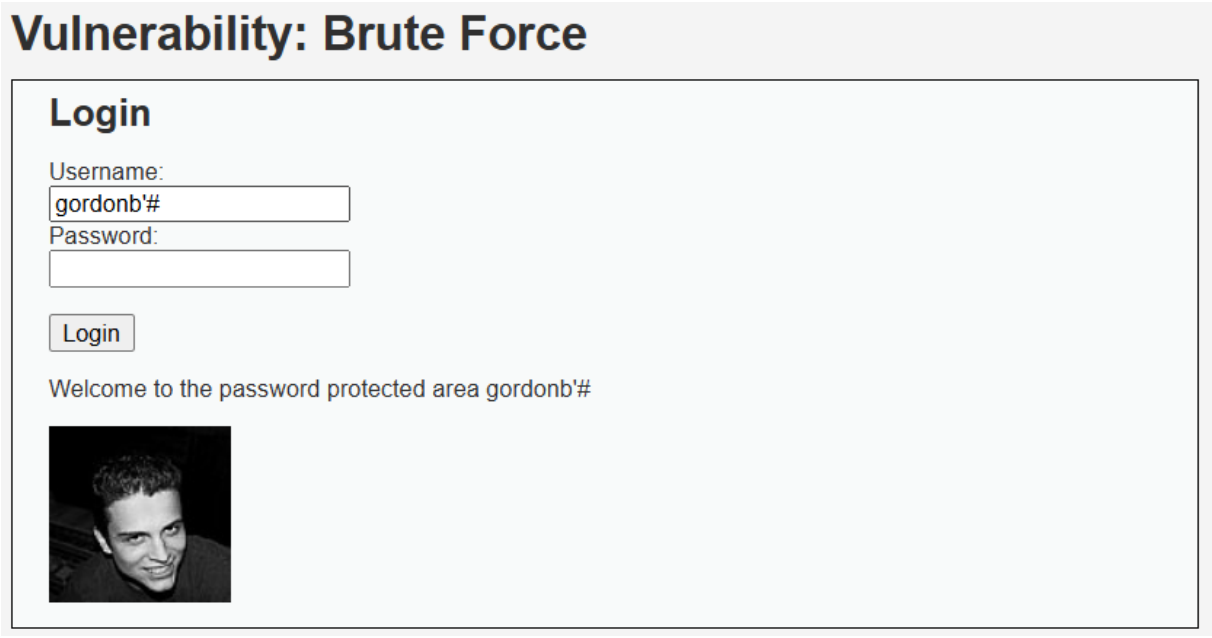
user = gordonb và password = abc123



Hình 3.10. Đăng nhập với user và password

Kịch bản 2: Kẻ tấn công chỉ khai thác được mỗi cột user của bảng users trong cơ sở dữ liệu hoặc biết cả user và password nhưng password đã được mã hóa một chiều (hashing) làm cho việc giải mã khó khăn hơn.

Kẻ tấn công vẫn có thể login bypass thành công khi biết mỗi user đã được giải thích ở chương 2.



Hình 3.11. Đăng nhập khi chỉ biết user

3.3. Phân tích kỹ thuật

Về mặt lý thuyết cơ sở dữ liệu, toán tử UNION được thiết kế nhằm kết hợp nhiều tập kết quả của các câu lệnh SELECT khác nhau thành một tập kết quả duy nhất. Cơ chế này được sử dụng hợp pháp trong các truy vấn phức tạp, tuy nhiên lại trở thành công cụ cực kỳ nguy hiểm khi bị lợi dụng trong tấn công SQL Injection [5], [7].

Cơ chế khai thác chi tiết

Cú pháp chuẩn của UNION:

```
SELECT column_A FROM table_1
UNION
SELECT column_B FROM table_2;
```

Trong cuộc tấn công, payload được đưa vào:

```
' UNION SELECT user, password FROM users #
```

Câu truy vấn ban đầu của hệ thống:

```
SELECT first_name, last_name FROM users WHERE user_id = '$id';
```

Sau khi bị chèn payload, câu lệnh tại máy chủ trở thành:

```
SELECT first_name, last_name FROM users
WHERE user_id = '' UNION SELECT user, password FROM users #';
```

Phân tích từng thành phần:

- Dấu ' đầu tiên kết thúc chuỗi mà lập trình viên tạo.
- Từ khóa UNION khởi tạo truy vấn mới do kẻ tấn công kiểm soát.
- Dấu # vô hiệu hóa phần còn lại của truy vấn gốc.
- Cơ sở dữ liệu không phân biệt truy vấn hợp lệ hay độc hại → thực thi toàn bộ câu lệnh.

Kết quả trả về được kết hợp giữa dữ liệu hợp lệ của hệ thống và dữ liệu nhạy cảm từ bảng users, khiến ứng dụng vô tình hiển thị thông tin bị đánh cắp trên giao diện người dùng.

Phân tích mức độ nguy hiểm

Union-based SQL Injection được đánh giá là một trong những kỹ thuật nguy hiểm nhất vì các lý do sau:

- Tính trực quan cao

- Không cần suy đoán như Blind SQL Injection.
- Dữ liệu được hiển thị trực tiếp → tốc độ tấn công cực nhanh.

- Tốc độ khai thác

Chỉ với vài truy vấn, kẻ tấn công có thể:

- Liệt kê toàn bộ bảng
- Liệt kê toàn bộ cột
- Trích xuất toàn bộ cơ sở dữ liệu → thực hiện Database Dump hoàn chỉnh.

Bỏ quan hoàn toàn cơ chế xác thực

Kẻ tấn công không cần đăng nhập vẫn thu được:

- Tài khoản admin
- Mật khẩu (dưới dạng hash)
- Thông tin nhạy cảm của người dùng

Hệ quả thực tế

Sau khi kiểm soát dữ liệu:

- Thay đổi nội dung hệ thống
- Cài mã độc, backdoor
- Leo thang đặc quyền

- Mở rộng tấn công sang các hệ thống khác

Đối với hệ thống doanh nghiệp, điều này có thể dẫn đến:

- Rò rỉ dữ liệu khách hàng
- Thiệt hại tài chính
- Mất uy tín thương hiệu
- Vi phạm pháp luật về bảo vệ dữ liệu cá nhân [5], [7].

Nhận định tổng hợp

Union-based SQL Injection không chỉ là một lỗi lập trình đơn giản, mà là lỗ hổng hệ thống nghiêm trọng phát sinh từ việc thiếu các nguyên tắc lập trình an toàn cơ bản:

- Không tách biệt dữ liệu và câu lệnh SQL
- Không sử dụng Prepared Statement
- Không kiểm soát dữ liệu đầu vào
- Không thiết kế quy trình phát triển phần mềm an toàn (Secure SDLC)

Trong môi trường thực tế, chỉ một lỗ hổng Union-based SQL Injection chưa được vá có thể dẫn đến sự sụp đổ toàn bộ hệ thống bảo mật của một ứng dụng web [4], [6], [7].

CHƯƠNG 4 – PHÂN TÍCH NGUYÊN NHÂN VÀ ĐỀ RA GIẢI PHÁP

4.1 Nguyên nhân dẫn tới lỗ hổng SQL Injection

Lỗ hổng SQL Injection không tự nhiên sinh ra từ nền tảng cơ sở dữ liệu, mà nó xuất phát từ sai sót trong quá trình phát triển ứng dụng và cấu hình hệ thống. Có thể chia làm các nguyên nhân chính sau.

Không kiểm soát dữ liệu đầu vào. Đây là nguyên nhân cơ bản nhất. Các lập trình viên thường mắc sai lầm khi tin tưởng tuyệt đối vào dữ liệu do người dùng nhập vào [7]:

- Vấn đề: Ứng dụng nhận dữ liệu từ các nguồn như form đăng nhập, tham số trên URL, cookie hoặc HTTP header mà không thực hiện bất kỳ bước kiểm tra, lọc bỏ ký tự đặc biệt hay xác thực kiểu dữ liệu nào.
- Hệ quả: Các ký tự điều khiển của ngôn ngữ SQL (như dấu nháy đơn ', dấu chấm phẩy ;, dấu gạch nối --) được phép đi qua và đi thẳng vào câu truy vấn.

Sử dụng cơ chế ghép chuỗi câu lệnh:

- Vấn đề: Thay vì sử dụng các cơ chế an toàn để truyền tham số, lập trình viên lại sử dụng phép cộng chuỗi (String Concatenation) để xây dựng câu lệnh SQL động [5].
- Việc ghép chuỗi này khiến trình biên dịch SQL không thể phân biệt được đâu là dữ liệu của người dùng. Khi kẻ tấn công chèn các từ khóa SQL vào, chúng sẽ được coi là một phần của mã lệnh thực thi.

Cấu hình hiển thị lỗi quá chi tiết

- Vấn đề: Các môi trường phát triển (Development) thường bật chế độ hiển thị lỗi chi tiết để debug. Tuy nhiên, nếu quên tắt chế độ này khi triển khai thực tế (Production), ứng dụng sẽ in toàn bộ thông báo lỗi của Database ra màn hình khi có sự cố [5].
- Hệ quả: Kẻ tấn công dựa vào thông báo lỗi để biết được loại cơ sở dữ liệu, phiên bản, và cấu trúc câu truy vấn bị lỗi để điều chỉnh payload tấn công.

4.2. Phương pháp phát hiện lỗ hổng

Việc phát hiện sớm lỗ hổng SQL Injection đóng vai trò quan trọng trong việc bảo vệ hệ thống thông tin. Các phương pháp phát hiện có thể chia thành hai nhóm chính: thủ công và tự động [7].

4.2.1. Thủ công

Phương pháp thủ công giúp người kiểm thử hiểu rõ hành vi của hệ thống và cơ chế xử lý dữ liệu đầu vào, từ đó phát hiện các dấu hiệu bất thường của lỗ hổng SQL Injection.

- Kiểm tra ký tự đặc biệt: Nhập ký tự dấu nháy đơn (') vào tất cả các điểm nhập liệu của ứng dụng (form đăng nhập, ô tìm kiếm, tham số URL, cookie…). Nếu hệ thống xuất hiện lỗi SQL hoặc phản hồi bất thường, đây là dấu hiệu phổ biến cho thấy dữ liệu đầu vào không được xử lý an toàn [7].
- Kiểm tra cú pháp truy vấn: Thử nghiệm với các biểu thức SQL cụ thể sao cho khi thay đổi một phần của dữ liệu đầu vào sẽ tạo ra các phản hồi khác nhau từ hệ thống. Việc so sánh phản hồi giữa các trường hợp này giúp xác định khả năng tồn tại của lỗ hổng.
- Kiểm tra điều kiện Boolean**:** Chèn các điều kiện logic luôn đúng và luôn sai, ví dụ: OR 1=1 hoặc OR 1=2

  Sau đó quan sát sự khác biệt trong phản hồi của ứng dụng. Nếu hệ thống phản hồi khác nhau tương ứng với các điều kiện này, rất có thể ứng dụng đang tồn tại lỗ hổng SQL Injection.

Phương pháp thủ công có ưu điểm là chính xác và linh hoạt, tuy nhiên đòi hỏi kiến thức chuyên môn cao và tốn nhiều thời gian thực hiện.

4.2.2. Tự động

Bên cạnh phương pháp thủ công, các công cụ quét bảo mật chuyên dụng giúp tự động hóa quá trình phát hiện lỗ hổng SQL Injection, cho phép kiểm tra toàn bộ ứng dụng một cách nhanh chóng và toàn diện.

Một số công cụ phổ biến:

- SQLMap
- Acunetix
- Burp Suite (Burp Scanner)
- OWASP ZAP

Các công cụ này có khả năng:

- Tự động phát hiện điểm yếu
- Khai thác thử nghiệm
- Phân tích mức độ nghiêm trọng của lỗ hổng
- Đề xuất phương án khắc phục

Việc kết hợp phương pháp thủ công và công cụ tự động giúp nâng cao độ chính xác và hiệu quả trong quá trình đánh giá bảo mật hệ thống [7].

4.3. Giải pháp phòng chống Prepared Statement

Prepared Statement được đánh giá là giải pháp hiệu quả và an toàn nhất trong việc phòng chống tấn công SQL Injection hiện nay. Khác với cơ chế xây dựng truy vấn truyền thống bằng cách ghép chuỗi, Prepared Statement đảm bảo tách biệt tuyệt đối giữa mã lệnh (Code) và dữ liệu (Data)**,** từ đó vô hiệu hóa hoàn toàn khả năng can thiệp của kẻ tấn công vào cấu trúc câu lệnh SQL [5], [6].

Cơ chế này hoạt động qua 2 bước:

1. Prepare: Ứng dụng gửi một mẫu câu lệnh SQL cố định đến máy chủ cơ sở dữ liệu. Ở giai đoạn này, câu lệnh chỉ chứa các vị trí tham số đại diện (placeholder) dưới dạng dấu ?. Hệ quản trị cơ sở dữ liệu sẽ tiến hành phân tích cú pháp, biên dịch và xây dựng kế hoạch thực thi (Execution Plan) cho câu lệnh này.
2. Execute: Ứng dụng gửi giá trị thực tế của người dùng vào. Database sẽ coi các giá trị này đơn thuần là dữ liệu văn bản, không bao giờ biên dịch lại. Do đó, dù hacker có nhập ' OR 1=1 -- thì nó cũng chỉ được hiểu là một chuỗi ký tự vô hại, không thể thay đổi cấu trúc logic của câu lệnh SQL [5], [6].

Minh họa khắc phục lỗ hổng trên mã nguồn Dựa trên đoạn mã lỗi đã phân tích ở Chương 3, nhóm đề xuất phương án sửa lỗi với Prepared Statement như sau:

```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["___

    // 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>";
```

Hình 4.1. Cấu hình gây ra lỗi.

```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 ) {
```

Hình 4.2. Cấu hình an toàn

Cơ chế bảo mật của Prepared Statement dựa trên nguyên tắc Pre-compilation (Biên dịch trước). Hệ quản trị cơ sở dữ liệu (DBMS) sẽ biên dịch cấu trúc câu lệnh SQL và xây dựng kế hoạch thực thi (Execution Plan) trước khi dữ liệu người dùng được đưa vào.

Nhờ đó, dữ liệu đầu vào (Input) dù có chứa các ký tự đặc biệt của SQL (như ', OR, DROP) thì cũng chỉ được hệ thống coi là chuỗi văn bản thuần túy (Literal values), không có khả năng thay đổi cấu trúc logic hoặc cú pháp của câu truy vấn gốc.

4.4. Đề xuất giải pháp phòng vệ khác

Bên cạnh việc sử dụng Prepared Statement để ngăn chặn SQL Injection ở mức mã nguồn, hệ thống cần được bảo vệ theo mô hình phòng thủ nhiều lớp (Defense in Depth) nhằm giảm thiểu rủi ro ngay cả khi một lớp bảo vệ bị vượt qua [5], [6].

4.4.1. Giải pháp triển khai tường lửa và hệ thống phát hiện, ngăn chặn xâm nhập (Firewall – IDS/IPS)

Trong các hệ thống mạng hiện nay, việc kết nối trực tiếp với Internet làm gia tăng nguy cơ bị tấn công từ bên ngoài thông qua các hình thức như quét cổng, khai thác lỗ hổng dịch vụ và tấn công từ chối dịch vụ. Theo nghiên cứu về an ninh mạng, tường lửa đóng vai trò là tuyến phòng thủ đầu tiên, trong khi IDS/IPS giúp phát hiện và phản ứng với các hành vi xâm nhập bất thường [5], [6].

Giải pháp được đề xuất là triển khai mô hình bảo mật gồm:

- Tường lửa để lọc lưu lượng theo chính sách bảo mật
- IDS/IPS để giám sát, phân tích gói tin và tự động cảnh báo hoặc chặn các hành vi đáng ngờ

4.4.2. Giải pháp mã hóa dữ liệu trong lưu trữ và truyền tải

Dữ liệu nhạy cảm là mục tiêu hàng đầu của các cuộc tấn công mạng. Theo tiêu chuẩn an toàn thông tin, dữ liệu cần được bảo vệ ngay cả khi hệ thống bị xâm nhập bằng cách áp dụng các cơ chế mã hóa phù hợp [2], [5].

Giải pháp bao gồm:

- Mã hóa dữ liệu khi lưu trữ trong cơ sở dữ liệu
- Mã hóa dữ liệu trong quá trình truyền tải thông qua các giao thức an toàn
- Phân loại dữ liệu để áp dụng mức độ mã hóa phù hợp

4.4.3. Giải pháp quản lý truy cập và xác thực đa yếu tố (MFA)

Nhiều nghiên cứu chỉ ra rằng các mối đe dọa nội bộ và việc lạm dụng quyền truy cập là nguyên nhân phổ biến gây mất an toàn thông tin. Việc quản lý truy cập không chặt chẽ làm gia tăng nguy cơ rò rỉ dữ liệu từ bên trong hệ thống [5], [7].

Giải pháp đề xuất:

- Áp dụng xác thực đa yếu tố (MFA)
- Phân quyền theo nguyên tắc đặc quyền tối thiểu
- Ghi nhận và giám sát nhật ký truy cập hệ thống

4.4.4. Nâng cao nhận thức và đào tạo an ninh mạng cho người dùng

Nhiều cuộc tấn công hiện nay khai thác yếu tố con người thay vì lỗ hổng kỹ thuật, đặc biệt là các hình thức **lừa đảo qua email (phishing)**, đánh cắp thông tin đăng nhập và cài mã độc thông qua thao tác người dùng [5], [6].

Việc tổ chức các chương trình đào tạo định kỳ về an ninh mạng giúp:

- Giảm đáng kể nguy cơ bị tấn công từ yếu tố con người
- Nâng cao hiệu quả của các giải pháp kỹ thuật đã triển khai
- Tăng khả năng phản ứng khi xảy ra sự cố an toàn thông tin

PHẦN 3 – KẾT LUẬN

Thông qua quá trình nghiên cứu lý thuyết kết hợp với thực nghiệm trên môi trường giả lập DVWA, báo cáo đã đạt được đầy đủ các mục tiêu nghiên cứu đề ra ban đầu.

Về phương diện lý thuyết**,** báo cáo đã hệ thống hóa và làm rõ bản chất của lỗ hổng SQL Injection, từ nguyên nhân hình thành, cơ chế hoạt động đến các dạng tấn công phổ biến. Đặc biệt, kỹ thuật Union-based SQL Injection được phân tích chi tiết như một trong những phương pháp khai thác nguy hiểm và hiệu quả nhất, cho phép kẻ tấn công trích xuất dữ liệu trái phép thông qua việc kết hợp kết quả truy vấn bằng toán tử UNION.

Về phương diện thực tiễn, nhóm đã mô phỏng thành công toàn bộ chuỗi tấn công trên môi trường DVWA, bao gồm các bước: xác định điểm yếu, dò số cột, thu thập thông tin hệ thống, liệt kê bảng – cột và cuối cùng là trích xuất toàn bộ bảng users. Kết quả thực nghiệm đã chứng minh rằng việc tin tưởng tuyệt đối vào dữ liệu đầu vào của người dùng **và** thiết kế truy vấn SQL không an toàn chính là nguyên nhân gốc rễ dẫn đến nguy cơ bị xâm nhập và mất an toàn thông tin nghiêm trọng.

Về phương diện giải pháp, báo cáo khẳng định Prepared Statement là biện pháp phòng chống SQL Injection hiệu quả và triệt để nhất hiện nay. Cơ chế tách biệt hoàn toàn giữa mã lệnh SQL và dữ liệu đầu vào giúp vô hiệu hóa hoàn toàn các nỗ lực tiêm nhiễm mã độc, vượt trội hơn so với các phương pháp lọc ký tự thủ công vốn dễ sai sót và khó duy trì. Bên cạnh đó, việc kết hợp Prepared Statement với các giải pháp phòng vệ nhiều lớp như WAF, mã hóa dữ liệu, quản lý truy cập và đào tạo nhận thức an ninh mạng đã tạo nên một hệ thống phòng thủ toàn diện.

Bài học kinh nghiệm

Từ kết quả nghiên cứu, có thể rút ra nhiều bài học quan trọng đối với quy trình phát triển phần mềm an toàn:

1. Nguyên tắc “Never Trust User Input”**:** Mọi dữ liệu từ người dùng, bao gồm URL, form, cookie, header HTTP, đều phải được coi là không an toàn và cần được kiểm soát chặt chẽ trước khi xử lý.

2. Tách biệt dữ liệu và mã lệnh: Việc sử dụng Prepared Statement hoặc cơ chế truy vấn tham số hóa là yêu cầu bắt buộc trong mọi ứng dụng web hiện đại.
3. Bảo mật theo chiều sâu (Defense in Depth)**:** Bảo mật không chỉ dừng lại ở mã nguồn mà phải được triển khai ở nhiều lớp: ứng dụng, cơ sở dữ liệu, mạng, người dùng và quy trình vận hành.
4. Kiểm thử bảo mật là một phần của phát triển phần mềm: An toàn thông tin phải được tích hợp xuyên suốt vòng đời phát triển hệ thống (Secure SDLC), không phải là bước bổ sung sau cùng.

Hướng phát triển của đề tài

Do giới hạn về thời gian và phạm vi nghiên cứu, báo cáo hiện tại mới tập trung chủ yếu vào kỹ thuật Union-based SQL Injection trên hệ quản trị MySQL. Trong tương lai, đề tài có thể được mở rộng theo các hướng sau:

- Nghiên cứu sâu hơn các biến thể phức tạp của SQL Injection như Blind SQL Injection (Time-based, Boolean-based), đặc biệt trong các hệ thống không hiển thị thông báo lỗi.
- Thực nghiệm trên các hệ quản trị cơ sở dữ liệu khác như SQL Server, PostgreSQL, Oracle để so sánh cơ chế khai thác và phòng vệ.
- Tích hợp các công cụ quét lỗ hổng tự động (SQLMap, Burp Suite, OWASP ZAP) vào quy trình DevSecOps, nhằm phát hiện và khắc phục lỗ hổng sớm ngay từ giai đoạn phát triển phần mềm.
- Mở rộng nghiên cứu sang các mô hình tấn công kết hợp nhiều kỹ thuật (chaining attacks) để đánh giá khả năng chịu lỗi và mức độ an toàn tổng thể của hệ thống.

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Nguyễn Khanh Văn (2014). *Giáo trình An toàn thông tin*, Nxb Bách Khoa Hà Nội.

[2] Trịnh Nhật Tiến (2014). *Giáo trình An toàn dữ liệu*, Nxb Đại học Quốc gia Hà Nội.

[3] Diễn đàn WhiteHat (2021). *SQL Injection - Khai thác Error Based SQL Injection*. Truy cập tại: https://whitehat.vn/threads/sql-injection-khai-thac-error-based-sql-injection.5664/#posts.

Tiếng Anh

[4] Anley, C. (2002). *Advanced SQL Injection In SQL Server Applications*, Next Generation Security Software Ltd.

[5] Clarke, J. (2012). *SQL Injection Attacks and Defense*, 2nd Edition, Syngress.

[6] OWASP (2021). "A03:2021 – Injection", *OWASP Top 10*. Truy cập tại: https://owasp.org/Top10/2021/A03_2021-Injection/ (Truy cập ngày 20/12/2025).

[7] Stuttard, D. & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*, 2nd Edition, John Wiley & Sons.