**BÁO CÁO BÀI THỰC HÀNH SỐ 9**

**KIỂM THỬ LỖ HỔNG XSS và CSRF**

**Họ và tên: Lưu Thy Thy**

**MSSV: 1050080078**

**Github:** [**https://github.com/ThyLuu/Bai-tap-ly-thuyet.git**](https://github.com/ThyLuu/Bai-tap-ly-thuyet.git)

**Youtube:** [**https://youtu.be/AIJyWonIg-g**](https://youtu.be/AIJyWonIg-g)

***Yêu cầu kết quả thực hành:***

Báo cáo kết quả thực hành có tên theo định dạng HoTenSV\_MSSV\_Lab06.docx

Trình bày quá trình kiểm thử theo mẫu sau

**Phần 1: Kiểm thử XSS**

Thực hiện kiểm thử lỗ hổng XSS trên website: chọn 1 trong các trang sau để thử nghiệm

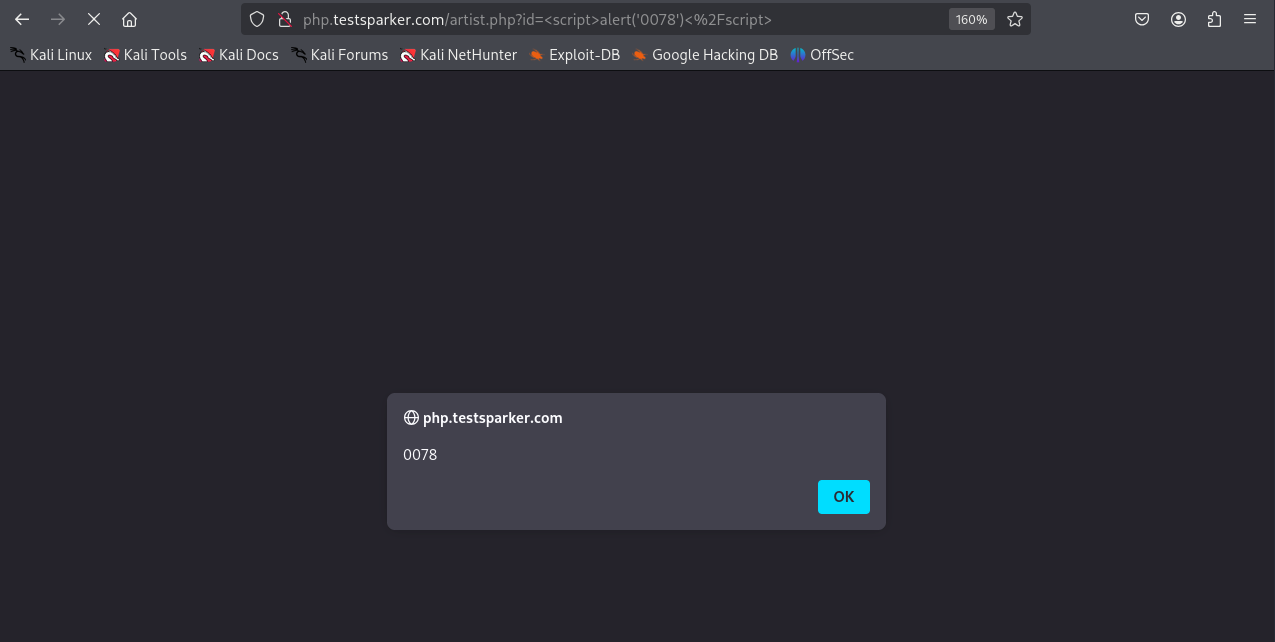
1. [http://demo.testfire.net](http://demo.testfire.net/)`
2. [http://php.testsparker.com](http://php.testsparker.com/)
3. [http://testasp.vulnweb.com](http://testasp.vulnweb.com/)
4. [http://testasp.vulnweb.com](http://testasp.vulnweb.com/)
5. [http://testphp.vulnweb.com](http://testphp.vulnweb.com/)
6. [http://www.webscantest.com](http://www.webscantest.com/)
7. [http://testhtml5.vulnweb.com](http://testhtml5.vulnweb.com/)
8. [http://aspnet.testsparker.com](http://aspnet.testsparker.com/)
9. [http://zero.webappsecurity.com](http://zero.webappsecurity.com/)

**Câu 1**

Thực hiện kiểm thử lỗ hổng XSS để thực thi đoạn mã hiển thị thông báo có chứa 4 số cuối MSSV. Hãy giải thích:



<script>alert(‘0078’)</script>



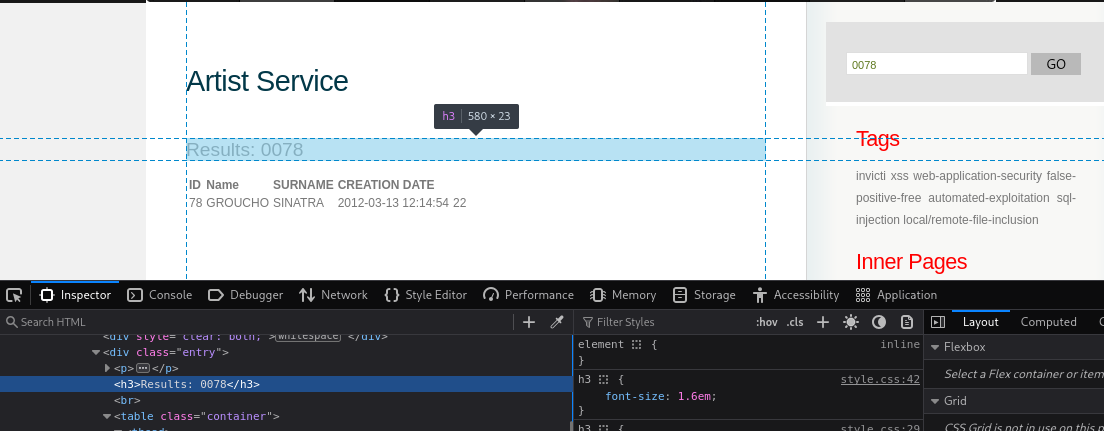
**-** Xác định các tham số đầu vào: **0.25 điểm**

* Phương thức GET
* Tham số đầu vào là id
* <http://php.testsparker.com/artist.php?id=0078>



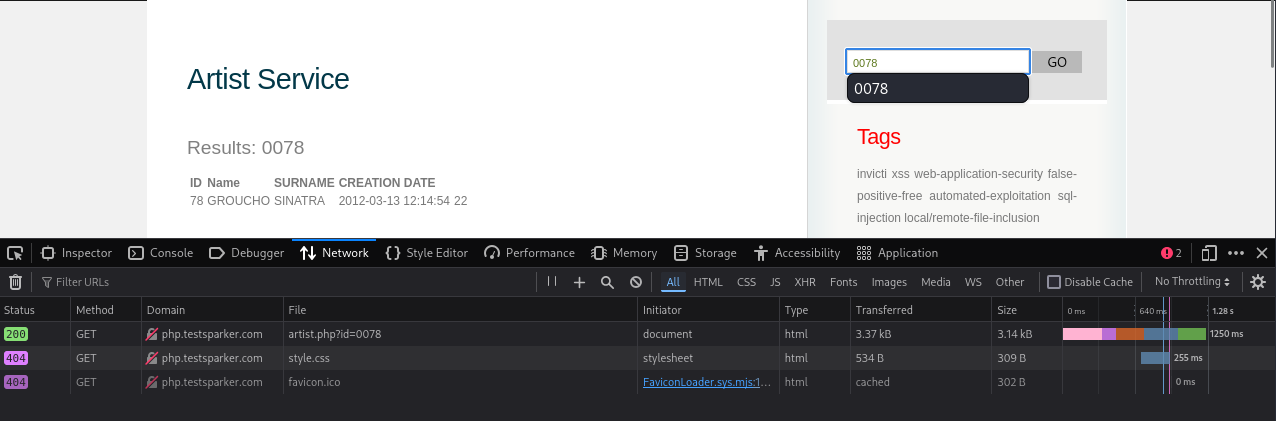
- Xác định giá trị đầu vào xuất hiện trong trang kết quả: **0.25 điểm**

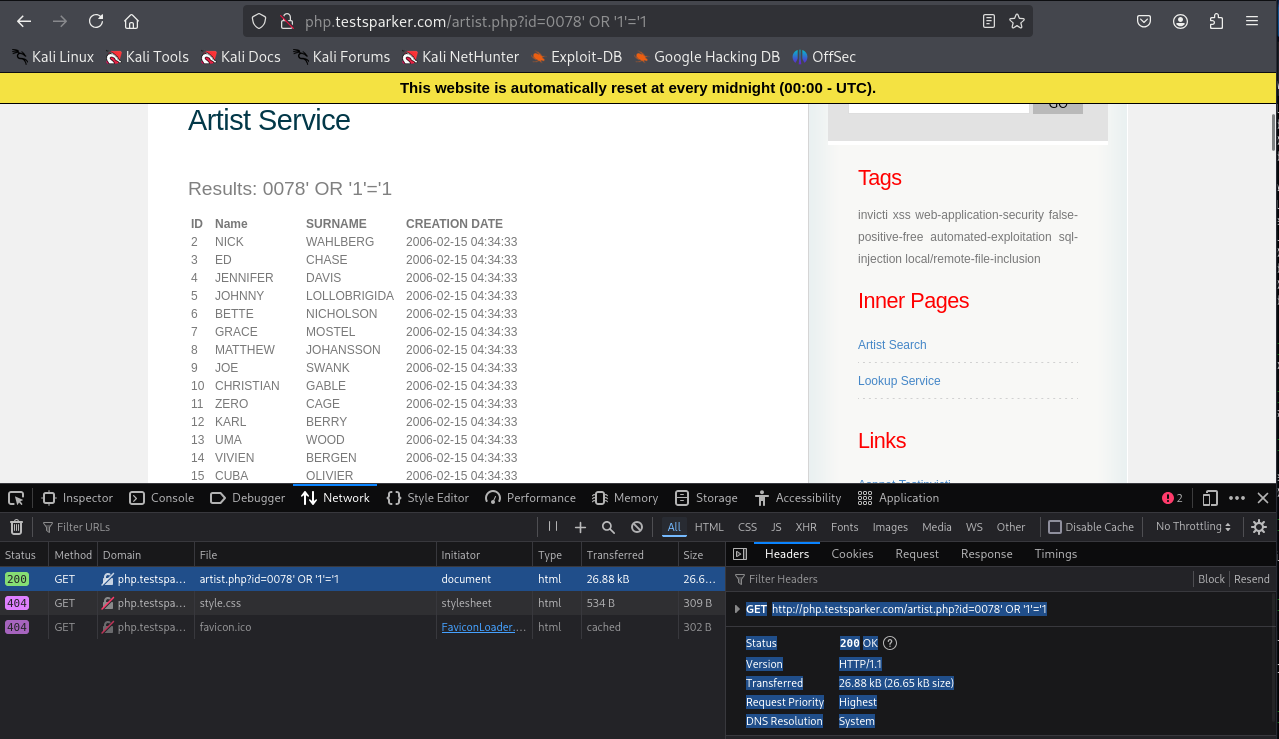
* Giá trị của id được chèn trực tiếp vào HTML, nằm trong thẻ tiêu đề hoặc một phần nội dung hiển thị tên nghệ sĩ, mà không qua hàm htmlspecialchars hay bất kỳ biện pháp mã hóa nào
* Khi truyền ?id=0078, trang sẽ hiển thị chữ “0078” y nguyên như text, chứng tỏ không có cơ chế lọc đầu vào

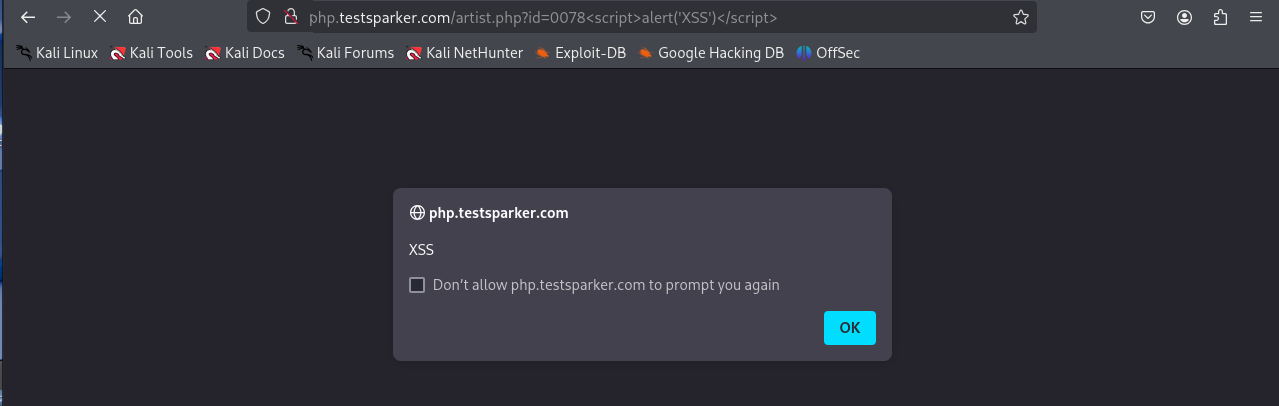


**-** Các bước kiểm thử và phán đoán: **0-1 điểm** theo các bước kiểm thử và logic trong phán đoán

* Trạng thái HTTP: Mã trạng thái 200 OK cho biết rằng yêu cầu GET đã được thực hiện thành công và máy chủ đã phản hồi bình thường. Đây là tín hiệu cho thấy trang hoặc tài nguyên yêu cầu có sẵn và không có lỗi.
* URL: http://php.testsparker.com/artist.php?id=0078 là một đường dẫn yêu cầu tài nguyên từ máy chủ với tham số id=0078. Hãy kiểm tra xem tham số này có thể bị tấn công hay không, ví dụ như kiểm thử với các giá trị đặc biệt hoặc bất thường như:
  + Các tham số có thể dẫn đến lỗi SQL Injection nếu máy chủ không xử lý đầu vào đúng cách.
  + Các tham số có thể thử nghiệm để kiểm tra khả năng bị tấn công XSS (Cross-Site Scripting).
* Có thể thử thay đổi giá trị của id như 0078' OR '1'='1, hoặc 0078<script>alert('XSS')</script>, để kiểm tra xem hệ thống có bảo vệ tốt chống lại các tấn công không.
  + Trạng thái HTTP 200 OK: Điều này có nghĩa là yêu cầu của bạn đã được xử lý thành công và máy chủ đã trả lại kết quả.
  + Dữ liệu trả về: Giao diện hiển thị danh sách các nghệ sĩ với ID, Tên, Họ và Ngày tạo. Dữ liệu trả về có vẻ như không bị giới hạn bởi tham số id, mà có thể là toàn bộ dữ liệu trong cơ sở dữ liệu.
  + SQL Injection Thành công: Việc sử dụng id=0078' OR '1'='1 có thể đã thay đổi truy vấn SQL gốc từ dạng: SELECT \* FROM artists WHERE id = '0078'; thành một truy vấn không hợp lệ nhưng vẫn đúng : SELECT \* FROM artists WHERE id = '0078' OR '1'='1';
  + Câu truy vấn này sẽ luôn trả về tất cả các bản ghi trong bảng artists vì điều kiện '1'='1' luôn đúng.







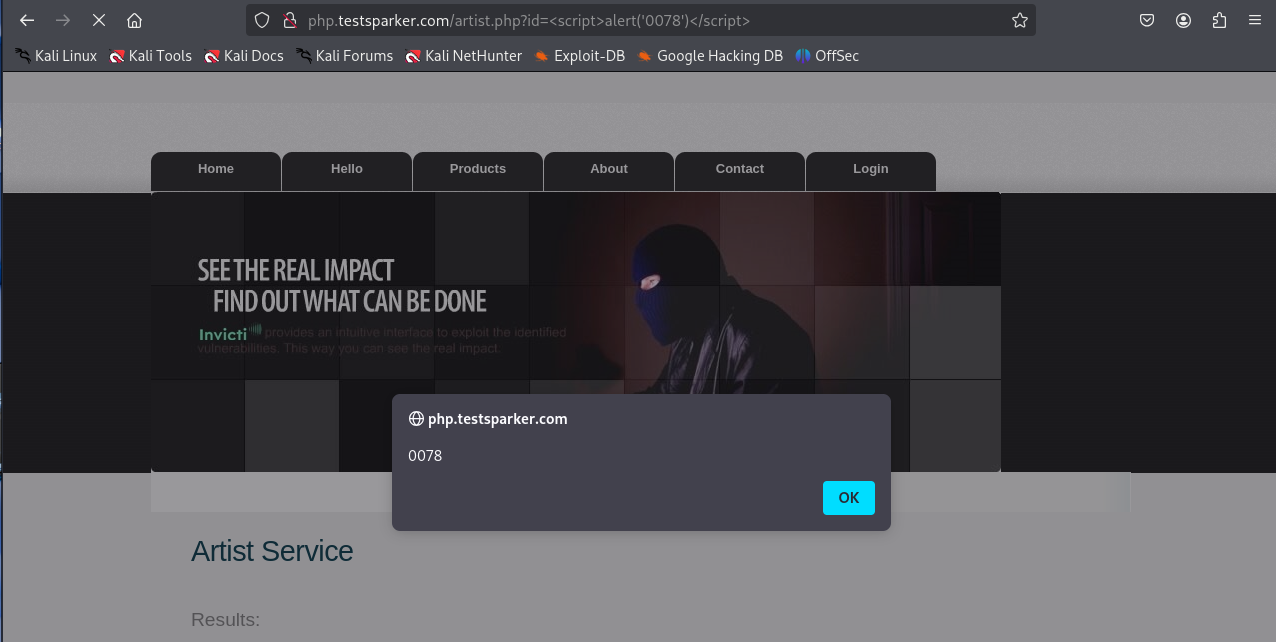
Kết quả :

* Hiển thị toàn bộ dữ liệu: Danh sách các nghệ sĩ được hiển thị, điều này cho thấy rằng ứng dụng không xử lý đầu vào đúng cách và không bảo vệ được chống lại SQL Injection.
* Rủi ro: Nếu ứng dụng tiếp tục để lỗ hổng SQL Injection tồn tại, một kẻ tấn công có thể sử dụng các mệnh đề SQL để truy cập, sửa đổi hoặc xóa dữ liệu trong cơ sở dữ liệu.
* Lỗ hổng SQL Injection: Đây là một vấn đề nghiêm trọng trong bảo mật ứng dụng web. Cần phải sửa lỗi này bằng cách sử dụng phương pháp bảo vệ như Prepared Statements hoặc ORMs để tránh việc chạy trực tiếp câu lệnh SQL với tham số người dùng.

- Giá trị kiểm thử cho thấy lỗ hổng và giải thích: **0.5 điểm**

* Để kiểm tra lỗ hổng XSS trên trang <http://php.testsparker.com/artist.php>, có thể sử dụng payload sau: <script>alert('0078')</script>
* [http://php.testsparker.com/artist.php?id=<script>alert('0078')</script](http://php.testsparker.com/artist.php?id=%3cscript%3ealert('0078')%3c/script)>





Trình duyệt sẽ hiển thị một hộp thoại thông báo với nội dung "0078". Điều này chứng tỏ rằng ứng dụng không thực hiện việc lọc hoặc mã hóa đầu vào từ tham số id trước khi hiển thị lên trang, dẫn đến lỗ hổng Reflected XSS.

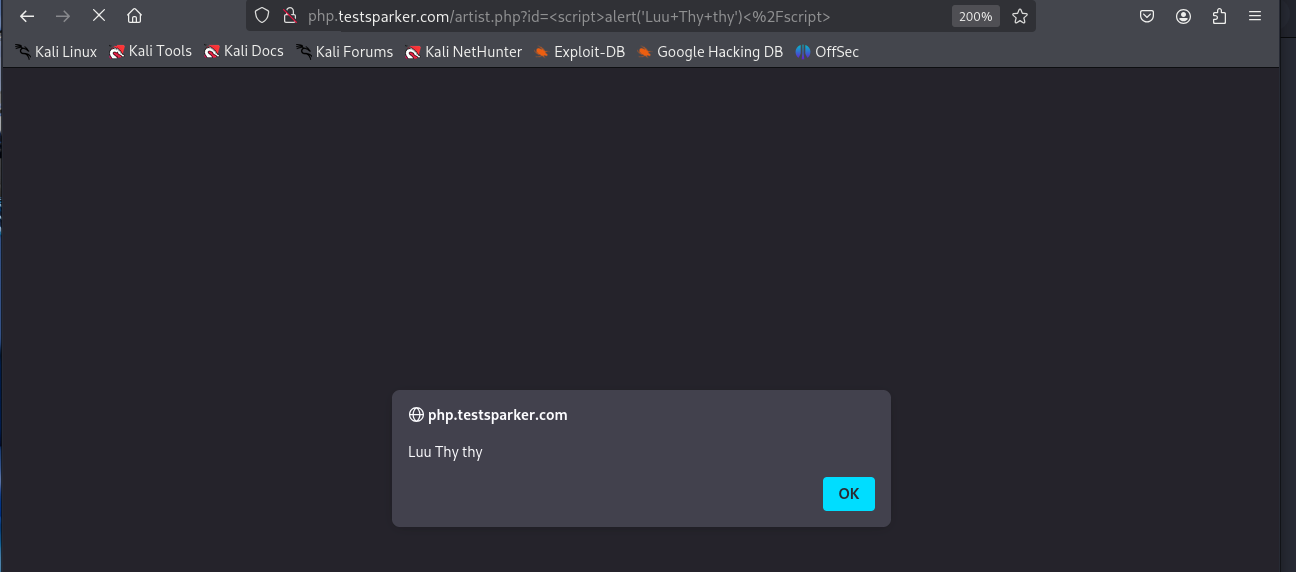
Giải thích :

* Cách thức hoạt động của lỗ hổng
  + Reflected XSS xảy ra khi dữ liệu từ người dùng được phản hồi trực tiếp trong trang web mà không được xử lý an toàn. Trong trường hợp này, giá trị của tham số id được chèn trực tiếp vào nội dung HTML mà không qua bất kỳ biện pháp lọc hoặc mã hóa nào.
  + Khi người dùng truy cập vào URL chứa mã JavaScript độc hại, trình duyệt sẽ thực thi mã đó, dẫn đến các hành vi không mong muốn như hiển thị thông báo, đánh cắp thông tin, hoặc thực hiện các hành động thay mặt người dùng.
* Tại sao payload <script>alert('0078')</script> lại hoạt động
  + Payload này chèn một thẻ <script> vào trang, và khi trình duyệt phân tích cú pháp HTML, nó sẽ thực thi nội dung bên trong thẻ <script>, cụ thể là hàm alert('0078').
  + Việc này chỉ xảy ra khi ứng dụng không thực hiện việc mã hóa các ký tự đặc biệt như <, >, và " trong đầu vào của người dùng, cho phép chèn mã HTML hoặc JavaScript tùy ý.
* Hậu quả tiềm ẩn: Kẻ tấn công có thể lợi dụng lỗ hổng này để thực hiện các hành vi như:
  + Đánh cắp thông tin đăng nhập: Bằng cách chèn mã JavaScript để gửi cookie hoặc thông tin người dùng đến máy chủ của kẻ tấn công.
  + Thực hiện hành động thay mặt người dùng: Như gửi yêu cầu đến máy chủ với quyền của người dùng hiện tại.
  + Phát tán phần mềm độc hại: Chèn mã để tải và thực thi phần mềm độc hại trên máy của người dùng.

**Câu 2**

Thực hiện kiểm thử lỗ hổng XSS để thực thi đoạn mã hiển thị thông báo có chứa tên của sinh viên. Điểm cho mỗi tham số đầu vào đã kiểm thử được:

<script>alert('Luu Thy thy')</script>



**-** Xác định các tham số đầu vào: **0.25 điểm**

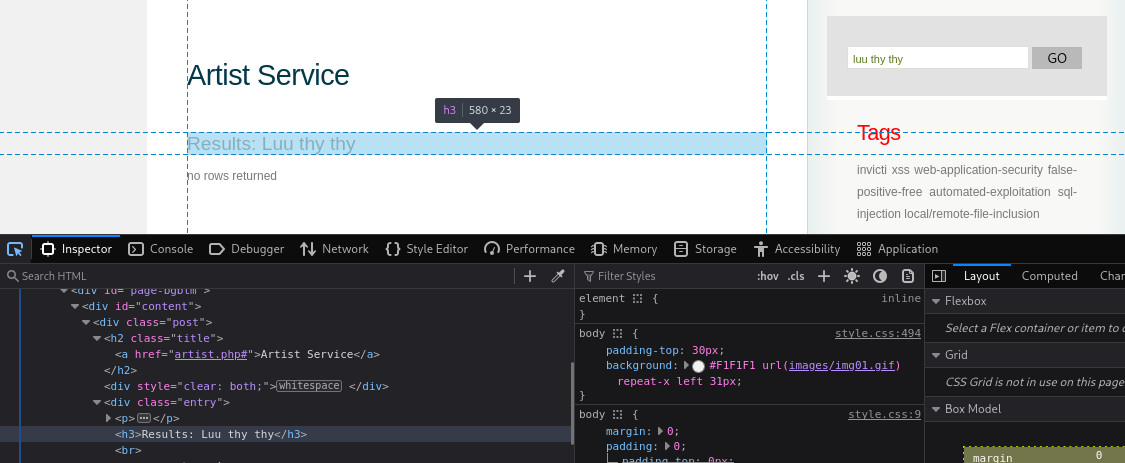
* Phương thức GET
* Tham số đầu vào: id
* http://php.testsparker.com/artist.php?id=Luu+thy+thy





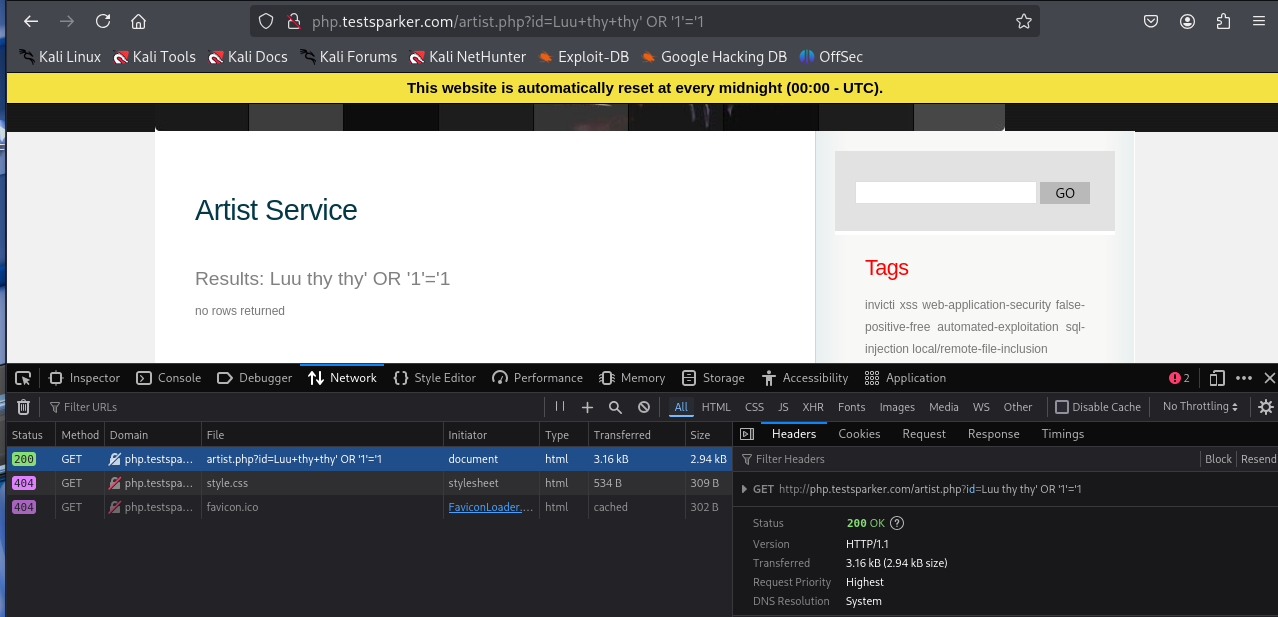
- Xác định giá trị đầu vào xuất hiện trong trang kết quả: **0.25 điểm**

* Giá trị của id được chèn trực tiếp vào HTML, nằm trong thẻ tiêu đề hoặc một phần nội dung hiển thị tên nghệ sĩ, mà không qua hàm htmlspecialchars hay bất kỳ biện pháp mã hóa nào
* Khi truyền ?id= Luu+thy+thy, trang sẽ hiển thị chữ “Luu thy thy” y nguyên như text, chứng tỏ không có cơ chế lọc đầu vào

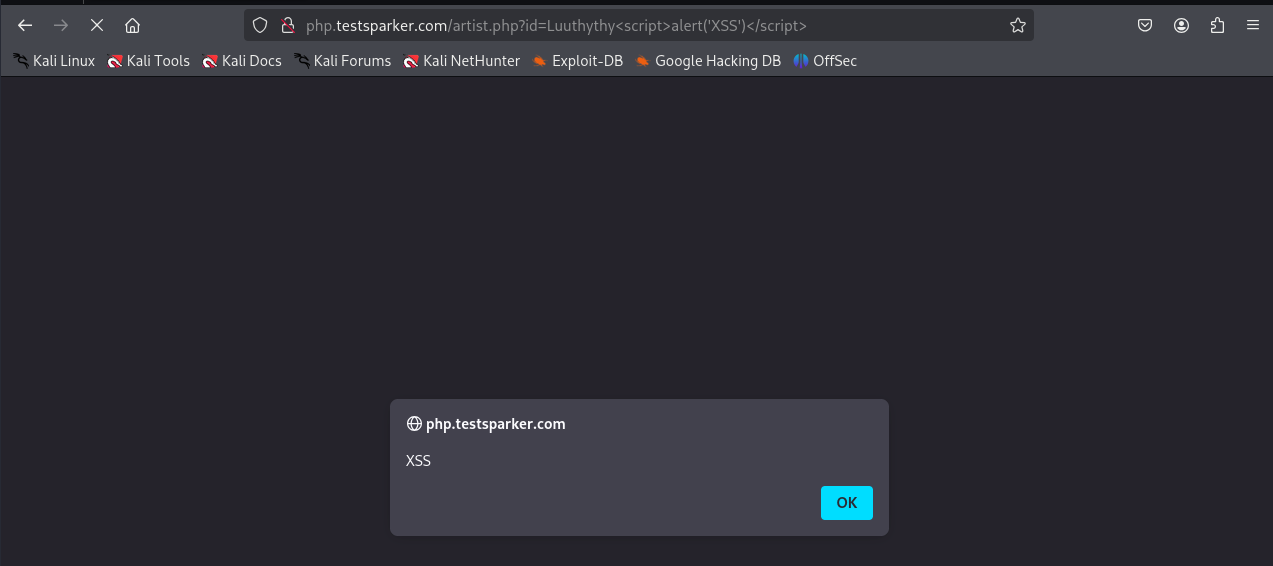


**-** Các bước kiểm thử và phán đoán: **0-1 điểm** theo các bước kiểm thử và logic trong phán đoán

* Trạng thái HTTP: Mã trạng thái 200 OK cho biết rằng yêu cầu GET đã được thực hiện thành công và máy chủ đã phản hồi bình thường. Đây là tín hiệu cho thấy trang hoặc tài nguyên yêu cầu có sẵn và không có lỗi.
* URL: http://php.testsparker.com/artist.php?id=Luu+thy+thy là một đường dẫn yêu cầu tài nguyên từ máy chủ với tham số id=Luu+thy+thy. Hãy kiểm tra xem tham số này có thể bị tấn công hay không, ví dụ như kiểm thử với các giá trị đặc biệt hoặc bất thường như:
  + Các tham số có thể dẫn đến lỗi SQL Injection nếu máy chủ không xử lý đầu vào đúng cách.
  + Các tham số có thể thử nghiệm để kiểm tra khả năng bị tấn công XSS (Cross-Site Scripting).
* Có thể thử thay đổi giá trị của id như Luu+thy+thy' OR '1'='1, hoặc Luuthythy<script>alert('XSS')</script>, để kiểm tra xem hệ thống có bảo vệ tốt chống lại các tấn công không.
  + Trạng thái HTTP 200 OK: Điều này có nghĩa là yêu cầu của bạn đã được xử lý thành công và máy chủ đã trả lại kết quả.
  + Phản hồi hiển thị: Results: Luu thy thy' OR '1'='1 không có dữ liệu nào được trả về.
  + So sánh với truy vấn id=0078' OR '1'='1
    - Kết quả: Cũng trả về 200 OK, nhưng hiển thị toàn bộ danh sách các bản ghi nghệ sĩ từ cơ sở dữ liệu.
    - Điều này chứng tỏ sự khác biệt trong cách hệ thống xử lý đầu vào chuỗi (Luu thy thy) so với số (0078)



<script>alert(‘0078’)</script>



* + Phán đoán
    - Trong truy vấn đầu (Luu thy thy' OR '1'='1), chuỗi ký tự có thể đã không được xử lý đúng cách về mặt cú pháp SQL hoặc bị escape một phần → khiến câu truy vấn bị lỗi logic và không thực thi mệnh đề OR đúng cách.
    - Ngược lại, truy vấn thứ hai hoạt động như mong đợi với SQL Injection, chứng minh hệ thống thực thi truy vấn SQL dựa trên đầu vào người dùng mà không xử lý bảo mật.

- Giá trị kiểm thử cho thấy lỗ hổng và giải thích: **0.5 điểm**

* http://php.testsparker.com/artist.php?id=Luu thy thy' OR '1'='1
* Truy vấn trả về:

Results: Luu thy thy' OR '1'='1

no rows returned

* Trong khi với truy vấn: http://php.testsparker.com/artist.php?id=0078' OR '1'='1 lại trả về toàn bộ danh sách nghệ sĩ, chứng tỏ câu truy vấn SQL đã bị "bẻ cong".
* Giải thích
  + Trường hợp "0078' OR '1'='1" (Dạng số):
    - Truy vấn có thể đã được xử lý như:

SELECT \* FROM artist WHERE id = '0078' OR '1'='1';

* + - * Vì '1'='1' luôn đúng, toàn bộ bản ghi được trả về
      * SQL Injection thành công.
    - Trường hợp "Luu thy thy' OR '1'='1" (Dạng chuỗi):
      * Truy vấn có thể đã thành:

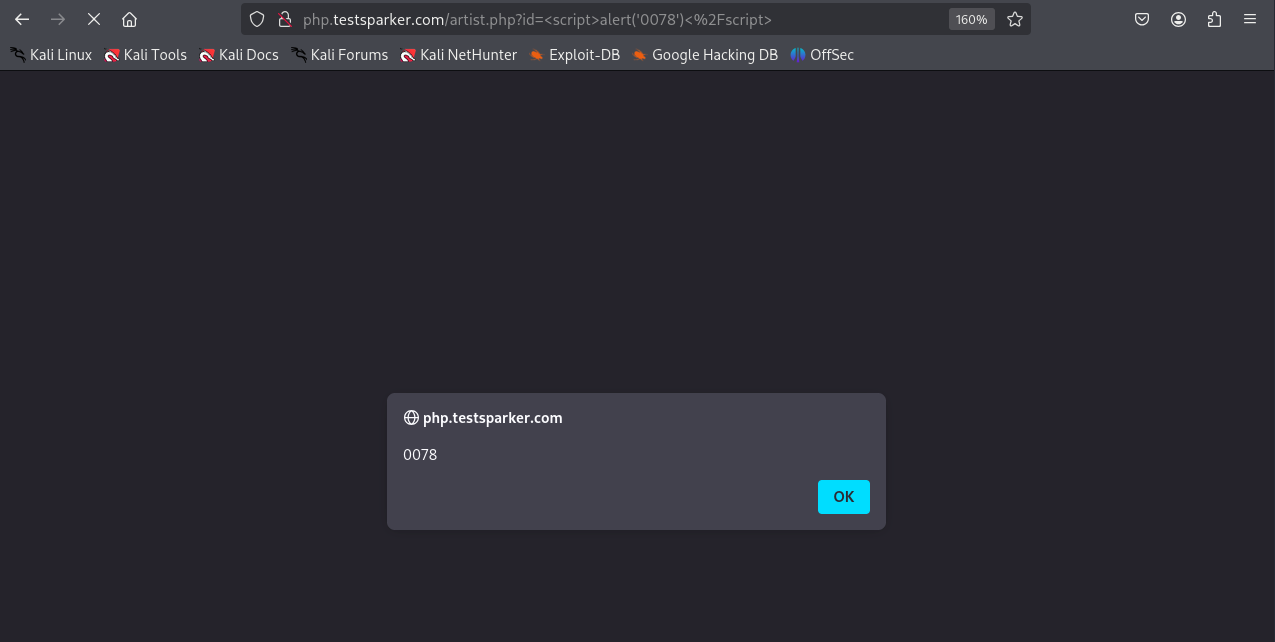
SELECT \* FROM artist WHERE id = 'Luu thy thy' OR '1'='1';

* + - * Tuy nhiên, nếu cột id trong database định dạng kiểu số, thì việc truyền chuỗi "Luu thy thy" có thể khiến câu truy vấn sai logic hoặc bị ép kiểu, dẫn đến việc WHERE không khớp bản ghi nào.
      * Ngoài ra, có thể ứng dụng đang xử lý chuỗi bằng một cách khác (escape chuỗi, casting dữ liệu), khiến mệnh đề OR '1'='1' không phát huy hiệu lực như mong muốn.

**Câu 3**

Thực hiện kiểm thử lỗ hổng XSS để thực thi đoạn mã hiển thị thông báo có chứa 4 số cuối MSSV. Hãy giải thích:





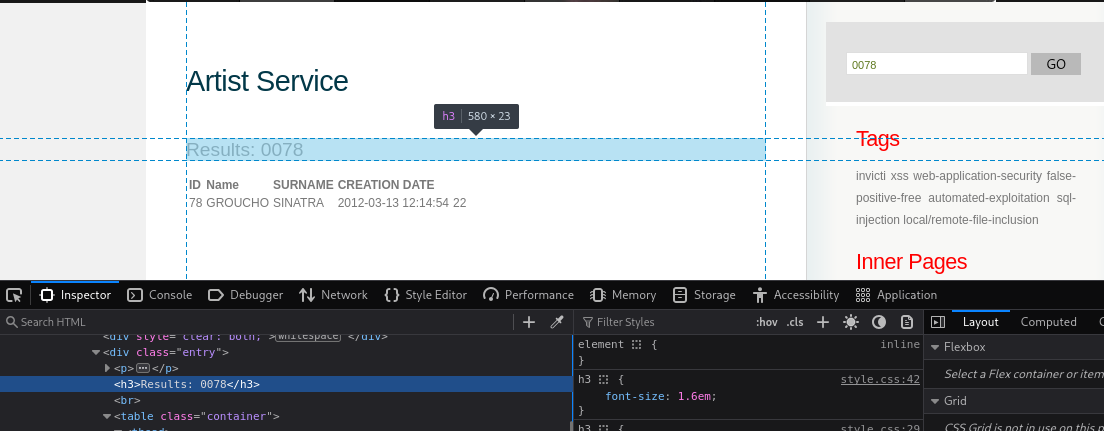
**-** Xác định các tham số đầu vào: **0.25 điểm**

* Phương thức GET
* Tham số đầu vào là id
* <http://php.testsparker.com/artist.php?id=0078>



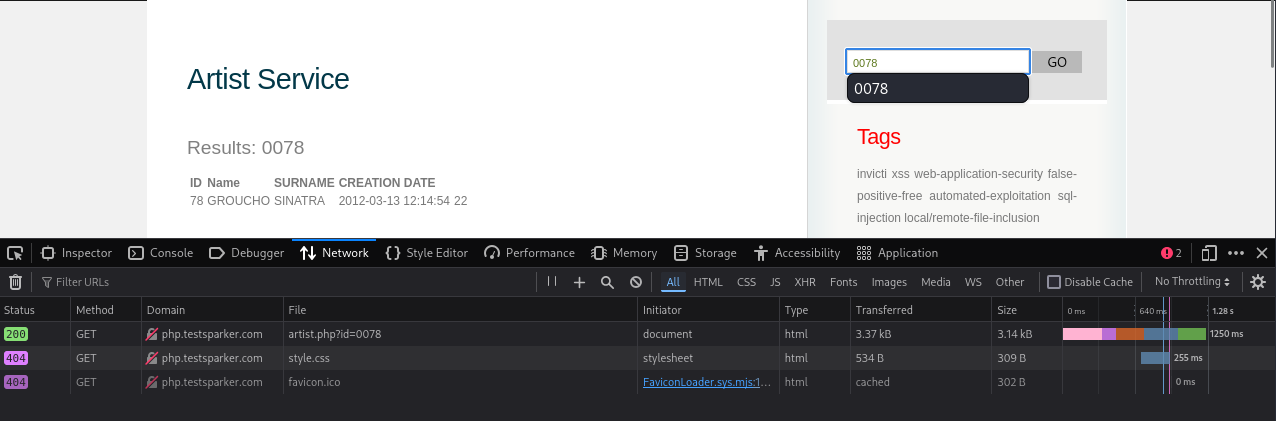
- Xác định giá trị đầu vào xuất hiện trong trang kết quả: **0.25 điểm**

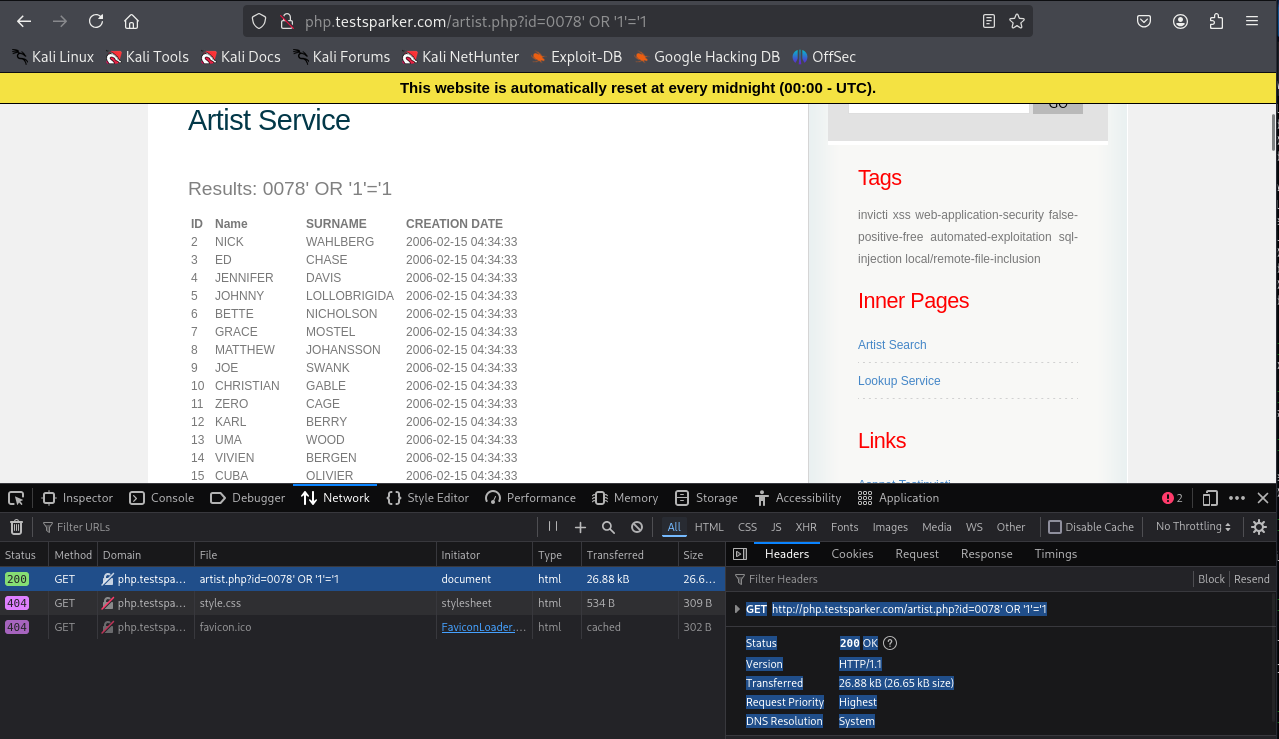
* Giá trị của id được chèn trực tiếp vào HTML, nằm trong thẻ tiêu đề hoặc một phần nội dung hiển thị tên nghệ sĩ, mà không qua hàm htmlspecialchars hay bất kỳ biện pháp mã hóa nào
* Khi truyền ?id=0078, trang sẽ hiển thị chữ “0078” y nguyên như text, chứng tỏ không có cơ chế lọc đầu vào

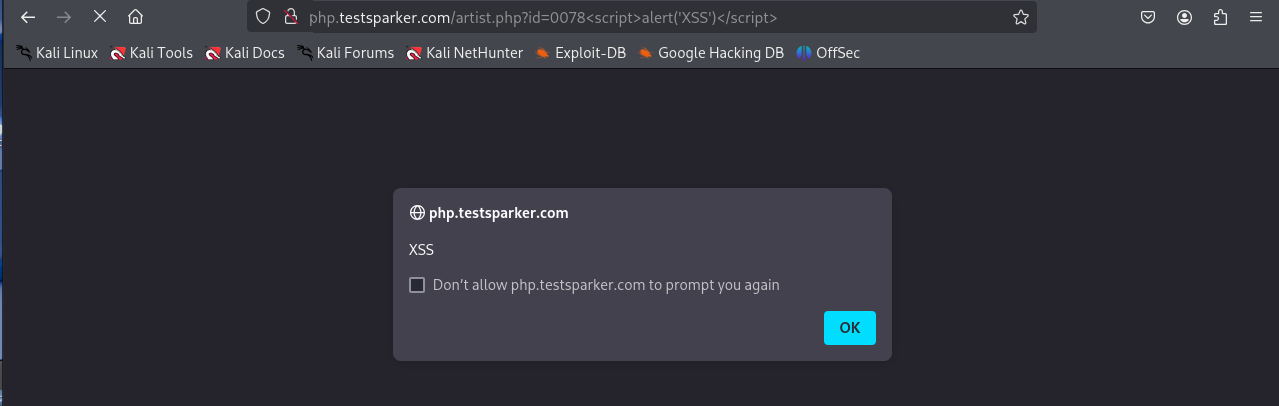


**-** Các bước kiểm thử và phán đoán: **0-1 điểm** theo các bước kiểm thử và logic trong phán đoán

* Trạng thái HTTP: Mã trạng thái 200 OK cho biết rằng yêu cầu GET đã được thực hiện thành công và máy chủ đã phản hồi bình thường. Đây là tín hiệu cho thấy trang hoặc tài nguyên yêu cầu có sẵn và không có lỗi.
* URL: http://php.testsparker.com/artist.php?id=0078 là một đường dẫn yêu cầu tài nguyên từ máy chủ với tham số id=0078. Hãy kiểm tra xem tham số này có thể bị tấn công hay không, ví dụ như kiểm thử với các giá trị đặc biệt hoặc bất thường như:
  + Các tham số có thể dẫn đến lỗi SQL Injection nếu máy chủ không xử lý đầu vào đúng cách.
  + Các tham số có thể thử nghiệm để kiểm tra khả năng bị tấn công XSS (Cross-Site Scripting).
* Có thể thử thay đổi giá trị của id như 0078' OR '1'='1, hoặc 0078<script>alert('XSS')</script>, để kiểm tra xem hệ thống có bảo vệ tốt chống lại các tấn công không.
  + Trạng thái HTTP 200 OK: Điều này có nghĩa là yêu cầu của bạn đã được xử lý thành công và máy chủ đã trả lại kết quả.
  + Dữ liệu trả về: Giao diện hiển thị danh sách các nghệ sĩ với ID, Tên, Họ và Ngày tạo. Dữ liệu trả về có vẻ như không bị giới hạn bởi tham số id, mà có thể là toàn bộ dữ liệu trong cơ sở dữ liệu.
  + SQL Injection Thành công: Việc sử dụng id=0078' OR '1'='1 có thể đã thay đổi truy vấn SQL gốc từ dạng: SELECT \* FROM artists WHERE id = '0078'; thành một truy vấn không hợp lệ nhưng vẫn đúng : SELECT \* FROM artists WHERE id = '0078' OR '1'='1';
  + Câu truy vấn này sẽ luôn trả về tất cả các bản ghi trong bảng artists vì điều kiện '1'='1' luôn đúng.







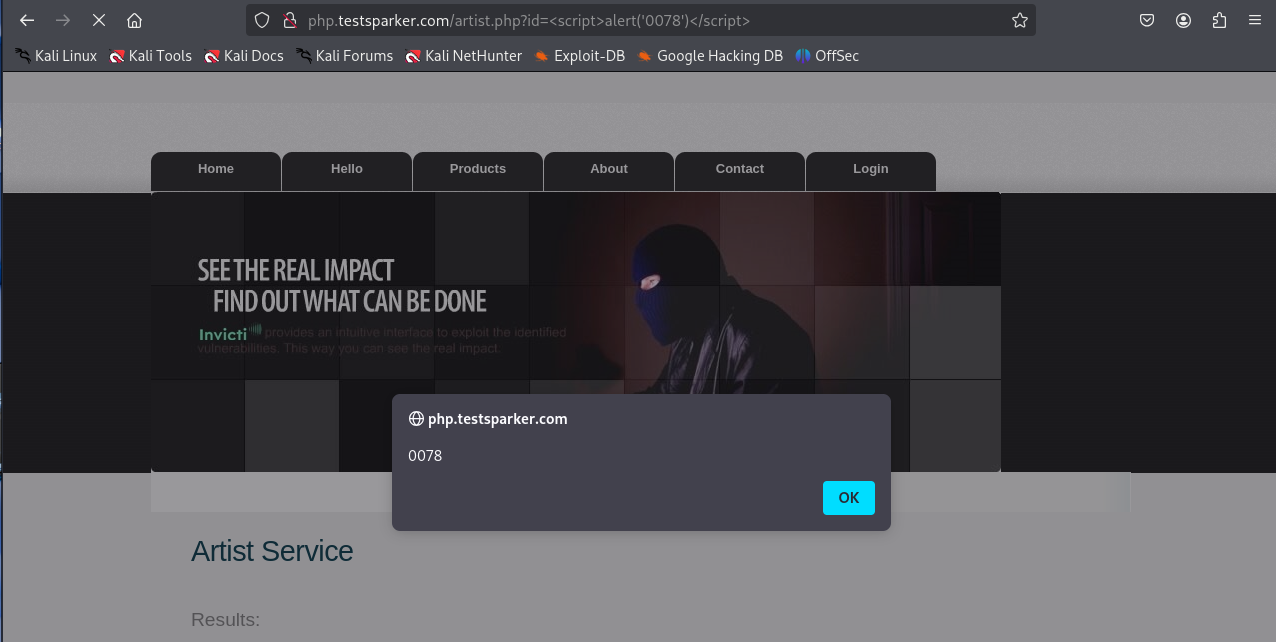
Kết quả :

* Hiển thị toàn bộ dữ liệu: Danh sách các nghệ sĩ được hiển thị, điều này cho thấy rằng ứng dụng không xử lý đầu vào đúng cách và không bảo vệ được chống lại SQL Injection.
* Rủi ro: Nếu ứng dụng tiếp tục để lỗ hổng SQL Injection tồn tại, một kẻ tấn công có thể sử dụng các mệnh đề SQL để truy cập, sửa đổi hoặc xóa dữ liệu trong cơ sở dữ liệu.
* Lỗ hổng SQL Injection: Đây là một vấn đề nghiêm trọng trong bảo mật ứng dụng web. Cần phải sửa lỗi này bằng cách sử dụng phương pháp bảo vệ như Prepared Statements hoặc ORMs để tránh việc chạy trực tiếp câu lệnh SQL với tham số người dùng.

- Giá trị kiểm thử cho thấy lỗ hổng và giải thích: **0.5 điểm**

* Để kiểm tra lỗ hổng XSS trên trang <http://php.testsparker.com/artist.php>, có thể sử dụng payload sau: <script>alert('0078')</script>
* [http://php.testsparker.com/artist.php?id=<script>alert('0078')</script](http://php.testsparker.com/artist.php?id=%3cscript%3ealert('0078')%3c/script)>





Trình duyệt sẽ hiển thị một hộp thoại thông báo với nội dung "0078". Điều này chứng tỏ rằng ứng dụng không thực hiện việc lọc hoặc mã hóa đầu vào từ tham số id trước khi hiển thị lên trang, dẫn đến lỗ hổng Reflected XSS.

Giải thích :

* Cách thức hoạt động của lỗ hổng
  + Reflected XSS xảy ra khi dữ liệu từ người dùng được phản hồi trực tiếp trong trang web mà không được xử lý an toàn. Trong trường hợp này, giá trị của tham số id được chèn trực tiếp vào nội dung HTML mà không qua bất kỳ biện pháp lọc hoặc mã hóa nào.
  + Khi người dùng truy cập vào URL chứa mã JavaScript độc hại, trình duyệt sẽ thực thi mã đó, dẫn đến các hành vi không mong muốn như hiển thị thông báo, đánh cắp thông tin, hoặc thực hiện các hành động thay mặt người dùng.
* Tại sao payload <script>alert('0078')</script> lại hoạt động
  + Payload này chèn một thẻ <script> vào trang, và khi trình duyệt phân tích cú pháp HTML, nó sẽ thực thi nội dung bên trong thẻ <script>, cụ thể là hàm alert('0078').
  + Việc này chỉ xảy ra khi ứng dụng không thực hiện việc mã hóa các ký tự đặc biệt như <, >, và " trong đầu vào của người dùng, cho phép chèn mã HTML hoặc JavaScript tùy ý.
* Hậu quả tiềm ẩn: Kẻ tấn công có thể lợi dụng lỗ hổng này để thực hiện các hành vi như:
  + Đánh cắp thông tin đăng nhập: Bằng cách chèn mã JavaScript để gửi cookie hoặc thông tin người dùng đến máy chủ của kẻ tấn công.
  + Thực hiện hành động thay mặt người dùng: Như gửi yêu cầu đến máy chủ với quyền của người dùng hiện tại.
  + Phát tán phần mềm độc hại: Chèn mã để tải và thực thi phần mềm độc hại trên máy của người dùng.

**Phần 2: Kiểm thử CSRF (2 điểm)**

Trình bày các bước thực hiện và kết quả kiểm thử lỗ hổng CSRF trên website chọn 1 trong các trang sau để thử nghiệm

1. [http://demo.testfire.net](http://demo.testfire.net/)
2. [http://php.testsparker.com](http://php.testsparker.com/)
3. [http://testasp.vulnweb.com](http://testasp.vulnweb.com/)
4. [http://testasp.vulnweb.com](http://testasp.vulnweb.com/)
5. [http://testphp.vulnweb.com](http://testphp.vulnweb.com/)
6. [http://www.webscantest.com](http://www.webscantest.com/)
7. [http://testhtml5.vulnweb.com](http://testhtml5.vulnweb.com/)
8. [http://aspnet.testsparker.com](http://aspnet.testsparker.com/)
9. [http://zero.webappsecurity.com](http://zero.webappsecurity.com/)