**Cross-site scripting: thực thi lệnh ở phía máy khách**

**Khái niệm:**

XSS là lỗ hổng web cho phép kẻ tấn công thâm nhập sự tương tác giữa người dùng và ứng dụng. Nó cho phép kẻ tấn công phá vỡ chính sách chung nguồn gốc, được thiết kế để tách biệt các trang web khác nhau.

XSS cho phép kẻ tấn công giả danh người dùng để thực hiện các hoạt động trái phép, truy cập dữ liệu nhạy cảm. Nếu nạn nhân là người có đặc quyền trong ứng dụng thì hắn có thể kiểm soát toàn bộ máy chủ.

**XSS hoạt động như thế nào?**

Nó sẽ tận dụng trang web có lỗ hổng để trả về đoạn mã Javascript độc hại cho người dùng. Khi đoạn mã đó được thực thi trong trình duyệt nạn nhân, kẻ tấn công có thể toàn quyền thâm nhập sự tương tác của họ với ứng dụng.

Bạn có thể kiểm tra XSS bằng việc tiêm 1 payload có thể kích hoạt Javascript bất kì. Cách phổ biến nhất là dùng hàm alert() vì nó ngắn, vô hại và rất dễ thành công.

Còn Chrome, bạn phải dùng print() thay thế.

**Các cách tấn công XSS:**

**Reflected XSS:**

**Khái niệm:**

Lỗ hổng này sinh ra khi ứng dụng nhận dữ liệu trong yêu cầu http và phản hồi theo 1 cách không an toàn.

Giả sử 1 trang web có chức năng tìm kiếm, tham số tìm kiếm là term trong url:

https://insecure-website.com/search?term=gift

Ứng dụng sẽ in tham số term trong phản hồi:

<p>You searched for: gift</p>

Giả sử ứng dụng không xử lí dữ liệu an toàn, kẻ tấn công có thể tiêm payload sau:

https://insecure-website.com/search?

term=<script>/\*+Bad+stuff+here...+\*/</script>

Url này sẽ hiển thị trong phản hồi:

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

Nếu người dùng khác yêu cầu url của kẻ tấn công, thì đoạn mã sẽ được thực thi trong trình duyệt của nạn nhân trong chính phiên làm việc của họ.

**Tác động của Reflected XSS:**

Nếu kẻ tấn công có thể kiểm soát đoạn mã được kích hoạt trong trình duyệt nạn nhân thì có thể tấn công người dùng đó theo những cách sau:

* Thực hiện những hành động trái phép
* Xem, điều chỉnh thông tin người dùng
* Tương tác với những người khác

Có rất nhiều cách để kẻ tấn công có thể bắt nạn nhân gửi yêu cầu họ kiểm soát. Nó bao gồm đặt những liên kết trên trang web được kiểm soát bởi kẻ tấn công hoặc trang web khác cho phép tạo nội dung hoặc gửi liên kết qua thư, tin nhắn,..

Vì cần kĩ thuật từ bên ngoài cho cuộc tấn công này nên ảnh hưởng của reflected xss sẽ không nghiêm trọng như stored.

**Làm sao để kiểm tra lỗ hổng Reflected XSS:**

Việc kiểm tra lỗ hổng Reflected XSS thủ công thường gồm những bước sau:

**Kiểm tra mọi đầu vào:** kiểm tra riêng những đầu vào trong yêu cầu http của ứng dụng bao gồm tham số và dữ liệu khác trong câu truy vấn url và nội dung tin, đường dẫn tập tin url kể cả tiêu đề http.

**Gửi những giá trị chữ số ngẫu nhiên:** với mỗi điểm vào, gửi đi giá trị ngẫu nhiên và xác định xem nó có trong phản hồi không. Giá trị này phải được thiết kế để bỏ qua lớp xác thực nên cần ngắn gọn và chỉ là những chữ số.

**Xác định bối cảnh reflection:** Nó có thể phản ánh giữa thẻ HTML, trong thuộc tính, chuỗi Javascript,..

**Kiểm tra payload**

**Thử tấn công trên trình duyệt**

**Cách phân biệt Stored và Reflected XSS:**

# Reflected XSS là lỗ hổng có thể thực thi Javascript trên Url (phương thức Get) .

# Stored XSS lưu trữ đầu vào lâu dài (phương thức Post)

# Bối cảnh là cách ứng dụng phản hồi đầu vào ở những vị trí khác nhau có thể ở thẻ Html, thuộc tính, chuỗi Javascript.

# Tùy thuộc vào bối cảnh khác nhau để đưa ra payload phù hợp.

**Stored XSS:**

**Khái niệm:**

Lỗ hổng này sinh ra khi ứng dụng nhận dữ liệu từ mã nguồn không đáng tin cậy và phản hồi theo cách không an toàn.

Giả sử 1 trang web cho phép người dùng gửi bình luận, được hiển thị với người dùng khác 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 đi, bất cứ ai xem bài viết cũng sẽ nhận được phản hồi sau từ ứng dụng:

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

Giả sử ứng dụng không xử lí dữ liệu đưa vào, kẻ tấn công có thể gửi đoạn mã độc sau:

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

Trong yêu cầu của kẻ tấn công bình luận sẽ được url mã hóa:

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

Bất cứ ai xem bài viết này sẽ nhận được phản hồi sau từ ứng dụng:

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

Đoạn mã sẽ được kích hoạt trong trình duyệt người dùng, trong phiên làm việc với ứng dụng.

**Tác động của Stored XSS:**

Sẽ giống với Reflected, có thể kiểm soát người dùng đó.

Điểm khác nhau là cách khai thác này sẽ được lưu trữ trong chính ứng dụng. Kẻ tấn công không cần bắt nạn nhân gửi yêu cầu đến máy chủ của mình.

Lỗ hổng XSS chỉ ảnh hưởng đến những người dùng đã đăng nhập. Nếu XSS là reflected, kẻ tấn công bắt buộc phải căn thời gian chuẩn: người dùng gửi yêu cầu của kẻ tấn công vào thời điểm anh ấy không đăng nhập sẽ không thâm nhập được. Trái lại khi XSS được lưu trữ, người dùng đã đăng nhập rồi mới bị tấn công.

**Làm sao để kiểm tra lỗ hổng Stored XSS?**

Bạn cần kiểm tra tất cả các điểm vào có liên quan mà qua đó dữ liệu mà kẻ tấn công kiểm soát có thể thâm nhập vào quá trình xử lí của ứng dụng và tất cả những điểm ra mà dữ liệu đó xuất hiện trong những phản hồi của ứng dụng.

**Những điểm vào trong xử lí của ứng dụng bao gồm:**

* Tham số hoặc dữ liệu khác trong chuỗi truy vấn url và nội dung tin nhắn
* Đường dẫn tập tin Url
* Những tiêu đề http
* Bất cứ trạm thu bên ngoài nào mà kẻ tấn công có thể đưa dữ liệu vào trong ứng dụng. Các trạm thu tồn tại phụ thuộc vào các chức năng được thực thi trong ứng dụng như ứng dụng webmail sẽ xử lý dữ liệu nhận được trong email; một ứng dụng cấp dữ liệu Twitter có thể xử lý dữ liệu có trong các tweet của bên thứ ba; và một công cụ tổng hợp tin tức sẽ bao gồm dữ liệu bắt nguồn từ các trang web khác.

Đầu ra cho lỗ hổng Stored XSS là những phản hồi trả về cho mọi người dùng ứng dụng.

Bước đầu tiên để kiểm tra lỗ hổng Stored XSS là đánh dấu những liên kết giữa điểm vào và điểm ra. Điều này rất khó vì:

* Dữ liệu được gửi đến bất cứ điểm vào nào cũng có thể được phát ra từ bất cứ điểm ra nào.
* Dữ liệu được lưu trữ bởi ứng dụng thường có lỗ hổng bị ghi đè vì những hành vi khác được thực hiện trong ứng dụng. Ví dụ, chức năng tìm kiếm có thể hiển thị danh sách tìm kiếm gần đây, nhanh chóng bị thay thế vì người dùng đang tìm kiếm cái khác.

Cách làm thực tế hơn là làm việc có hệ thống với những đầu vào dữ liệu, gửi đi 1 giá trị cụ thể và theo dõi phản hồi của ứng dụng để xác định giá trị gửi đi nằm ở đâu. Nên chú ý những chức năng của ứng dụng như bình luận trên blog posts. Khi quan sát thấy giá trị được gửi đi trong phản hồi, bạn cần xác định xem dữ liệu có được lưu trữ qua những yêu cầu khác không, vì nó có thể chỉ đơn giản là bị phản ánh tức thời.

Khi bạn đã xác định những liên kết giữa điểm vào và điểm ra trong quá trình xử lí của ứng dụng, mỗi liên kết cần được kiểm tra cụ thể để phát hiện stored xss.

**Xác định bối cảnh Stored XSS**

**Kiểm tra payload**

**Thử tấn công trên trình duyệt**

**DOM-based XSS:**

**Khái niệm:**

Lỗ hổng này sinh ra khi Javascript lấy dữ liệu từ source mà kẻ tấn công kiểm soát được như Url và gửi nó đến sink để thực thi code như eval() hoặc innerHTML. Nó cho phép kẻ tấn công đánh cắp tài khoản của người dùng khác.

Để tấn công DOM-based XSS, bạn cần đặt dữ liệu trong source để đưa đến sink thực thi code Javasciprt.

Source phổ biến nhất của DOM XSS là url, được truy suất qua đối tượng window.location. Một kẻ tấn công có thể tạo 1 liên kết gửi nạn nhân đến trang có lỗ hổng với 1 payload trong chuỗi truy vấn và phân đoạn của Url.

**Làm sao để kiểm tra DOM-based XSS:**

**Kiểm tra HTML sinks:**

Để kiểm tra DOM XSS trong HTML sink, đặt 1 chuỗi ngẫu nhiên trong source như location.search, sau đó sử dụng công cụ phát triển để phân tích HTML và tìm nơi chuỗi xuất hiện.

Lưu ý: chức năng “View source” sẽ không kiểm tra được DOM XSS được vì nó không tính đến những thay đổi đã được Javascript thực hiện trong HTML.

Với mỗi vị trí mà chuỗi bạn xuất hiện trong DOM, bạn cần xác định bối cảnh để đưa vào input phù hợp. Ví dụ, nếu chuỗi bạn xuất hiện trong 1 thuộc tính “…” hãy thử tiêm “ để thoát thuộc tính cũ.

Lưu ý trình duyệt sẽ xử lí khác nhau liên quan đến mã hóa url. Chrome, Firefox và Safari sẽ mã hóa url location.search và location.hash, còn IE11 và Microsoft Edge thì không. Nếu dữ liệu của bạn bị mã hóa trước khi xử lí thì cuộc tấn công XSS sẽ không hoạt động.

**Kiểm tra Javascript sinks:**

1. Với những sinks này, đầu vào của bạn ko cần thiết xuất hiện trong Dom, vì vậy bạn không cần tìm nó. Thay vào đó, bạn phải dùng Javascript debugger để xác định xem đầu vào của bạn đã gửi đến sink như thế nào.
2. Với mỗi source tiềm năng như location, bạn phải tìm kiếm tất cả những mã Javascript của tất cả trang chứa source đó.
3. Sau đó, bạn có thể dùng Javascript debugger để tạo 1 điểm dừng và theo dõi giá trị của source đc dùng thế nào.
4. Bạn có thể thấy source đó được gán đến biến khác. Nếu gặp trường hợp đó, bạn sẽ cần dùng chức năng tìm kiếm để theo dõi các biến này và xem liệu chúng có được gửi đến sink không.
5. Khi bạn thấy 1 sink được gửi dữ liệu có nguồn gốc từ 1 source bạn có thể dùng debugger để điều tra giá trị đó bằng cách di chuột vào biến để hiển thị giá trị của nó trước khi gửi đến sink.
6. Cuối cùng, bạn cần chỉnh sửa lại input để tấn công XSS thành công.

**Khai thác DOM XSS với những sources và sinks khác nhau:**

Về nguyên tắc, 1 trang web dính lỗ hổng Dom XSS nếu có đường dẫn thực thi mà qua đó dữ liệu có thể truyền từ source đến sink.

Trong thực tế, những sources và sinks khác nhau sẽ khác nhau về thuộc tính và hành vi nên cách khai thác sẽ bị ảnh hưởng.

Thêm nữa, đoạn mã của trang web có thể thực hiện xác thực hoặc xử lí dữ liệu khi cố gắng khai thác lỗ hổng.

Sink document.write sẽ làm việc với thẻ script, vì vậy bạn có thể dùng payload sau:

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

**Chú ý:** nội dung mà document.write viết sẽ có một số bối cảnh khác nhau buộc chúng ta phải khai thác khác đi. Ví dụ, bạn có thể cần đóng thẻ hiện tại trước payload Javascript.

Sink innerHTML không chấp nhận thẻ script trên trình duyệt hiện đại, cũng như thẻ svg sự kiện onload kích hoạt. Điều đó có nghĩa bạn sẽ cần sử dụng thẻ img hay iframe thay thế. Sự kiện onload hay onerror có thể được dùng chung. Ví dụ:

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

**Sources và sinks phụ thuộc bên thứ ba:**

Rất nhiều ứng dụng web hiện đại được dựng lên bằng thư viện của bên thứ ba, cung cấp nhiều hàm và thuộc tính có sẵn. Việc nhớ những sources và sinks tiềm năng để tấn công Dom XSS rất quan trọng.

**Dom XSS trong jQuery:**

Nếu thư viện Javascript như jQuery được dùng, hãy tìm kiếm những sinks có thể thay thế các thẻ trong Dom trên trang.

Ví dụ, hàm attr() có thể thay đổi thuộc tính của thẻ. Nếu dữ liệu được lấy từ source người dùng kiểm soát đc như url, sau đó gửi đến hàm attr(), có thể tận dụng giá trị đó để tấn công XSS.

Ở đây ta có Javascript thay đổi thuộc tính của thẻ href bằng việc sử dụng dữ liệu từ url:

$(function() {

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

});

Bạn có thể khai thác nó bằng việc điều chỉnh url để location.search chứa Javascript độc hại. Sau đó, nhấp vào back link sẽ thực thi:

?returnUrl=javascript:alert(document.domain)

Hãy chú ý đến hàm selector $() của jQuery, có thể được dùng để tiêm đối tượng độc hại trong Dom.

jQuery được dùng khá phổ biến, và lỗ hổng Dom XSS được gây ra bởi những trang web dùng selector này kết hợp với source location.hash để thêm hiệu ứng hay tự động cuộn những thành phần trong 1 trang. Hành vi này được thực thi bằng sự kiện hashchange, tương tự như sau:

$(window).on('hashchange', function() {

var element = $(location.hash);

element[0].scrollIntoView();

});

Khi hash được người dùng kiểm soảt, kẻ tấn công có thể tiêm XSS trong sink $(). Gần đây jQuery đã vá lỗ hổng này bằng việc ngăn chặn bạn khỏi tiêm html trong selector khi đầu vào bắt đầu bằng hash(#). Tuy nhiên, trong thực tế bạn vẫn tìm thấy đoạn mã có lỗ hổng.

Để khai thác lỗ hổng này, bạn cần tìm cách kích hoạt sự kiện hashchange không cần người dùng tương tác. Một trong những cách đơn giản nhất là dùng 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 có lỗ hổng với giá trị hash rỗng. Khi iframe được tải, XSS được thêm vào hash làm cho sự kiện hashchange kích hoạt.

Thậm chí những phiên bản mới nhất của jQuery vẫn có lỗ hổng với sink $(), miễn là bạn có toàn quyền kiểm soát đầu vào từ source, nó không yêu cầu tiền tố #.

**DOM XSS trong Angular JS:**

Nếu 1 thư viện như Angular JS được dùng, nó có thể thực thi Javascript không cần <, > hoặc sự kiện. Khi 1 trang web dùng thuộc tính ng-app trong thẻ HTML, nó sẽ được xử lí bởi Angular JS. Trong trường hợp này, Angular JS sẽ thực thi Javascript trong dấu {{}} có thể xảy ra trực tiếp trong HTML hoặc thuộc tính.

Payload: {{constructor.constructor('alert(1)')()}}

**DOM XSS kết hợp với dữ liệu được phản ảnh và lưu trữ:**

Một vài lỗ hổng DOM-based thuần khiết chỉ khép kín trong 1 trang. Nếu đoạn mã đọc dữ liệu từ url và viết nó vào sink nguy hiểm thì lỗ hổng hoàn toàn nằm ở phía máy khách.

Tuy nhiên source không chỉ giới hạn những dữ liệu được hiển thị bởi trình duyệt mà nó còn có nguồn gốc từ ứng dụng. Ví dụ, những trang web thường phản ánh những tham số của url trong phản hồi từ phía máy chủ. Đây chỉ là XSS bình thường nhưng nó cũng có thể dẫn đến reflected DOM XSS.

Trong lỗ hổng reflected DOM XSS, máy chủ xử lí dữ liệu từ yêu cầu và xuất dữ liệu ra. Dữ liệu bị phản ánh sẽ được đặt trong đoạn mã Javascript, hoặc 1 đối tượng trong Dom. Đoạn mã trên trang sau đó sẽ xử lí dữ liệu được phản ánh theo cách không an toàn, sẽ viết nó vào 1 sink nguy hiểm.

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

Những trang web cũng lưu trữ dữ liệu trên máy chủ và phản ánh nó ở bất cứ nơi đâu. Trong lỗ hổng stored Dom XSS, máy chủ nhận dữ liệu từ yêu cầu, lưu trữ nó và sẽ bao gồm dữ liệu trong những phản hồi sau này. Đoạn mã trong phản hồi sau này chứa sink xử lí dữ liệu không an toàn.

element.innerHTML = comment.author

**Khai thác lỗ hổng XSS:**

**Đánh cắp cookies:**

Đánh cắp cookies là 1 cách truyền thống để khai thác XSS. Đa số ứng dụng web dùng cookies để xử lí phiên. Bạn có thể khai thác lỗ hổng này để gửi cookies của nạn nhân đến tên miền của mình sau đó tiêm cookies vào trình duyệt và giả danh nạn nhân.

Trong thực tế, cách tiếp cận này có những giới hạn:

* Nạn nhân có thể không đăng nhập
* Rất nhiều ứng dụng giấu cookies của chúng khỏi Javascript bằng HttpOnly
* Phiên làm việc có thể thêm các yếu tố như địa chỉ Ip của người dùng
* Phiên có thể hết hạn trước khi bạn đánh cắp nó.

**Thu thập mật khẩu:**

Những ngày gần đây, rất nhiều người dùng hệ thống quản lí mật khẩu để tự đồng điền mật khẩu. Bạn có thể lợi dụng điều này bằng việc tạo ra mục nhập mật khẩu, đọc mật khẩu tự điền và gửi nó đến tên miền của mình. Kĩ thuật này tránh đa số vấn đề liên quan đến đánh cắp cookies, thậm chí có thể truy cập vào những tài khoản khác mà nạn nhận sử dụng lại mật khẩu này.

**Thực hiện CSRF:**

Bất cứ thứ gì người dùng hợp lệ làm được thì XSS cũng làm được. Phụ thuộc vào trang web bạn nhắm đến, bạn có thể bắt nạn nhân gửi tin nhắn, chấp nhận lời mời kết bạn hoặc chuyển 1 vài Bitcoin.

Một vài trang web thường cho người dùng đã đăng nhập thay đổi địa chỉ email không cần nhập lại mật khẩu. Nếu bạn thấy lỗ hổng XSS, bạn có thể bắt nó thay đổi địa chỉ email thành cái mình kiểm soát, sau đó đổi lại mật khẩu để đăng nhập vào tài khoản.

Loại tấn công này liên quan đến CSRF thì khá khó hiểu vì CSRF là 1 lỗ hổng độc lập. Khi CSRF độc lập nó có thể bị ngăn chặn bằng việc sử dụng mã anti-CSRF nhưng sẽ không đảm bảo nếu XSS cũng tồn tại.

**Thực thi đoạn mã ở máy khách ở những bối cảnh khác nhau:**

Khi kiểm tra reflected hay stored XSS, nhiệm vụ chính là xác định bối cảnh của XSS:

* Vị trí xuất hiện dữ liệu do người dùng kiểm soát trong phản hồi
* Xác thực đầu vào hay quá trình xử lí dữ liệu khác được thực hiện bởi ứng dụng.

**XSS giữa các thẻ HTML:**

Khi dữ liệu do người dùng kiểm soát được trả về giữa các thẻ html, bạn cần 1 vài thẻ html mới để thực thi đoạn mã Javascript.

Ví dụ:

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

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

**XSS trong thuộc tính HTML:**

Khi bối cảnh XSS nằm trong thuộc tính của HTML, bạn có thể phải hủy giá trị thuộc tính đó, đóng thẻ và làm một cái mới. Ví dụ:

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

Rất nhiều trường hợp dấu “>” bị chặn hoặc mã hóa, vì vậy đầu vào của bạn không thể đóng thẻ. Để hủy giá trị của thuộc tính, bạn có thể thêm thuộc tính mới như các sự kiện

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

Payload ở trên tạo 1 sự kiện onfocus sẽ thực thi Javascript nếu thẻ nhận được focus. autofocus để tự động focus. Cuối cùng thêm x=” để đóng thẻ phù hợp.

Thỉnh thoảng bạn có thể kích hoạt đoạn mã Javascript không cần đóng giá trị của thuộc tính. Ví dụ, nếu thuộc tính là href, bạn có thể sử dụng giao thức javascript giả để thực thi.

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

Một số trang web mã hóa angle brackets, để hủy giá trị của thuộc tính cũ không nhất thiết phải thêm thuộc tính là những sự kiện tự động. Bạn có thể dùng thẻ canonical.

Bạn có thể khai thác hành vi này bằng cách dùng access keys và sự tương tác của người dùng trên Chrome. Access keys cho phép bạn cung cấp tổ hợp phím để tham chiếu đến thẻ chỉ định. Thuộc tính accesskey cho phép bạn khai báo kí tự kết hợp sẽ làm cho các sự kiện được kích hoạt.

XSS trong khai thác đầu vào bị ẩn:

<input type="hidden" accesskey="X" onclick="alert(1)">

Trên Chrome:

<link rel="canonical" accesskey="X" onclick="alert(1)" />

**XSS trong Javascript:**

**Việc hủy các đoạn mã hiện tại:**

Rất dễ để đóng đoạn mã script cũ trong Javascript và mở thẻ html mới để kích hoạt Javascript khác.

Ví dụ, nếu bối cảnh của XSS như sau:

<script>

...

var input = 'controllable data here';

...

</script>

Sau đó bạn có thể dùng payload sau để thoát ra khỏi Javascript hiện có và thực thi mã của riêng bạn:

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

Lí do nó hoạt động là đầu tiên trình duyệt phân tích HTML để xác định các thành phần trong trang có chứa script, và sau đó mới phân tích Javascript để hiểu và thực thi code. Payload ở trên làm cho script gốc bị hỏng vì chuỗi chưa kết thúc. Nhưng nó không ngăn chặn script khác được thực thi.

**Thoát ra khỏi chuỗi Javascript:**

Bạn có thể thoát ra khỏi chuỗi và thực thi code Javascript trực tiếp. Việc chỉnh sửa đoạn script theo bối cảnh XSS rất quan trọng vì cú pháp sai sẽ ngăn chặn toàn bộ đoạn mã được thực thi.

Một vài cách hữu dụng để đóng chuỗi:

'-alert(document.domain)-'

';alert(document.domain)//

Một vài ứng dụng ngăn chặn đầu vào khỏi việc đóng chuỗi Javascript bằng việc thêm \ trước dấu nháy đơn.

Ví dụ, giả sử đầu vào là: ';alert(document.domain)//

sẽ chuyển đổi thành \';alert(document.domain)//

Bạn có thể sử dụng payload sau thay thế: \';alert(document.domain)//

sẽ chuyển đổi thành \\';alert(document.domain)//

Ở đây, dấu \ đầu tiên sẽ làm dấu \ thứ hai thành kí tự bình thường nên đóng chuỗi được 🡪 Thành công

Một vài trang web rất khó tấn công XSS vì bị giới hạn một số kí tự. Bạn cần thử nhiều cách khác nhau để vượt qua được kĩ thuật bảo mật này.

Một cách làm là dùng câu lệnh throw để xử lí exception. Nó cho phép bạn gửi đối số đến hàm mà không dùng ngoặc đơn (). Đoạn mã sau gán alert cho onerror và câu lệnh throw sẽ gửi 1 đến alert. Kết quả là hàm alert(1).

onerror=alert;throw 1

**Sử dụng mã hóa HTML:**

Khi bối cảnh XSS là code Javascript trong thuộc tính của thẻ, như event handler, có thể mã hóa Html để bỏ qua bộ lọc.

Khi trình duyệt phân tích thẻ Html và thuộc tính trong phản hồi, nó sẽ giải mã các giá trị của thuộc tính trước khi được xử lí thêm. Nếu máy chủ chặn hoặc sàng lọc những kí tự để khai thác XSS, bạn có thể vượt qua dễ dàng.

Ví dụ, bối cảnh XSS như sau:

<a href="#" onclick="... var input='controllable data here'; ...">

và ứng dụng sẽ chặn hoặc thêm \ trước dấu ‘, bạn có thể dùng payload sau:

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

&apos; là thực thể Html đại diện cho dấu nháy đơn. Vì trình duyệt sẽ giải mã HTML giá trị của thuộc tính của onclick trước khi thực thi Javascript nên tấn công thành công.

**XSS trong khuôn mẫu Javascript:**

Khuôn mẫu Javascript là những chuỗi cho phép nhúng biểu thức Javascript. Biểu thức nhúng sẽ được đánh giá và nối vào văn bản xung quanh. Biểu thức nhúng có cú pháp là: ${...}

Ví dụ, đoạn mã sau sẽ in thông báo chào mừng bao gồm tên người dùng:

document.getElementById('message').innerText =

`Welcome, ${user.displayName}.`;

Khi bối cảnh XSS là trong khuôn mẫu Javascript, không cần thiết phải thoát chuỗi, bạn chỉ cần sử dụng {…} để nhúng biểu thức Javascript, nó sẽ đc thực thi khi mẫu được xử lí.

Ví dụ nếu bối cảnh XSS như sau:

<script>

...

var input = `controllable data here`;

...

</script>

Sau đó bạn có thể dùng payload sau để thực thi Javascript mà không cần thoát khỏi chuỗi:

${alert(document.domain)}