**Cách để ngăn chặn XSS:**

Ngăn chặn XSS có thể thực hiện qua hai lớp phòng thủ sau:

* Mã hóa dữ liệu ở đầu ra
* Xác thực đầu vào

**Mã hóa dữ liệu ở đầu ra:**

Mã hóa dữ liệu do người dùng kiểm soát trước khi kết xuất 1 trang web.

Bối cảnh của XSS giúp xác định loại mã hóa bạn cần dùng:

Trong bối cảnh của HTML, bạn nên chuyển những giá trị không được phép thành thực thể HTML:

**< 🡪 &lt;**

**> 🡪 &gt;**

Trong chuỗi Javascript, những kí tự không phải chữ, số nên được mã hóa Unicode:

**< 🡪 \u003c**

**> 🡪 \u003e**

Thỉnh thoảng bạn sẽ cần dùng nhiều lớp mã hóa. Ví dụ để nhúng đầu vào trong event handlers, bạn cần xử lí với hai bối cảnh Javascript và Html. Vì vậy bạn sẽ cần mã hóa Unicode đầu vào trước, sau đó mới mã hóa HTML.

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

**Xác thực đầu vào ở điểm đến:**

Mã hóa có thể là lớp phòng thủ quan trọng nhất của XSS, nhưng nó không đủ để ngăn chặn lỗ hổng XSS trong mọi bối cảnh. Bạn nên xác thực đầu vào 1 cách nghiêm ngặt nhất khi người dùng nhận được.

Ví dụ:

* Nếu người dùng gửi 1 Url sẽ được trả về trong phản hồi, xác thực xem nó có bắt đầu với giao thức an toàn như http và https không. Ngược lại, một vài người có thể khai thác trang web của bạn với giao thức gây hại như javascipt hay data.
* Nếu người dùng cung cấp 1 giá trị phải là số, xác thực xem nó có chứa số nguyên không.
* Xác thực đầu vào chỉ chứa những kí tự cho phép.

Việc xác thực đầu vào nên chặn được những kí tự không hợp lệ. Một cách tiếp cận khác, cố gắng dọn sạch đầu vào không hợp lệ sẽ dễ bị lỗi hơn và nên tránh.

**Whiltelisting vs Blacklisting:**

Việc xác thực đầu vào nên dùng whitelists hơn blacklists. Ví dụ, thay vì cố gắng tạo 1 danh sách những giao thức độc hại (javascript, data), chỉ cần tạo 1 danh sách những giao thức an toàn là được (http, https) và chặn những giao thức khác.

Nó sẽ đảm bảo lớp phòng thủ của bạn không bị vỡ khi những giao thức độc hại mới xuất hiện và ít bị tấn công hơn khi tìm cách làm xáo trộn các giá trị không hợp lệ để trốn tránh danh sách đen.

**Cho phép HTML an toàn:**

Việc cho phép người dùng gửi HTML nên tránh, nhưng thỉnh thoảng nó rất cần thiết. Ví dụ, 1 trang blog có thể cho phép những bình luận chứa thẻ HTML.

Cách tiếp cận kinh điển là cố gắng lọc ra những thẻ và Javascript độc hại. Bạn có thể thực thi nó bằng cách sử dụng những thẻ và javascipt được phép, nhưng nhờ sự khác biệt của các công cụ phân tích trình duyệt và XSS đột biến, cách tiếp cận này thường rất khó để bảo mật.

Lựa chọn tệ nhất là sử dụng thư viện Javascript cho phép lọc và mã hóa trong trình duyệt người dùng, như DOMPurify. Nhiều thư viện khác cho phép người dùng cung cấp nội dung theo định dạng đánh dấu và chuyển nó về HTML. Không may mắn, tất cả thư viện này đều dính lỗi XSS, vì vậy nó không phải là lựa chọn hoàn hảo. Nếu bạn dùng chúng, phải thường xuyên theo dõi những bản cập nhật về bảo mật.

Lưu ý: Ngoài Javascript, một số nội dung khác như CSS và thậm chí HTML có thể độc hại trong 1 số tình huống.

**Làm sao để ngăn chặn XSS bằng cách sử dụng công cụ có sẵn:**

Rất nhiều trang web hiện đại sử dụng những công cụ có sẵn ở phía máy chủ như Twig và Freemarker để nhúng nội dung động trong HTML. Chúng thường dùng hệ thống escape riêng. Ví dụ, trong Twig, bạn có thể dùng bộ lọc e(), với đối số sẽ khai báo bối cảnh sau:

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

Một vài công cụ có sẵn khác như Jinja và React, escape nội dung động bằng cách ngăn chặn đa số lỗi XSS

Chúng ta sẽ đánh giá lại những tính năng escape rõ ràng hơn khi bạn cân nhắc sử dụng công cụ hay thư viện có sẵn.

Nếu bạn trực tiếp nối đầu vào trong chuỗi có sẵn, bạn sẽ bị dính lỗ hổng server-side template injection, thường nghiêm trọng hơn XSS.

**Làm sao để ngăn chặn XSS trong PHP:**

Trong PHP, có những hàm dựng sẵn để mã hóa những thực thể như htmlentities. Bạn nên gọi hàm này khi escape đầu vào trong bối cảnh HTML. Hàm này nên được gọi với ba đối số sau:

* Đầu vào
* ENT-QUOTES xác định tất cả quotes nên được mã hóa.
* Bộ kí tự, đa số trường hợp nên là Utf-8.

Ví dụ:

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

Khi trong bối cảnh Javascript, bạn cần escape Unicode như đã được mô tả. Không may mắn, Php không cung cấp Api để Unicode-escape 1 chuỗi. Đây là một vài đoạn mã làm điều đó trong PHP:

<?php

function 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>

Thay vào đó, bạn có thể dùng công cụ có sẵn.

**Làm sao để ngăn chặn XSS phía máy khách trong Javascript:**

Để escape đầu vào trong bối cảnh HTML trong Javascript, bạn cần bộ mã hóa HTML của riêng mình vì Javascript không cung cấp API để mã hóa HTML. Đây là một vài đoạn mã Javascript chuyển đổi 1 chuỗi sang thực thể HTML:

function htmlEncode(str){

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

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

});

}

Sau đó, bạn có thể dùng hàm này như sau:

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

Nếu đầu vào của bạn trong 1 chuỗi Javascript, bạn cần 1 bộ mã hóa thực hiện escape Unicode. Đây là đoạn mã 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 có thể dùng hàm này như sau:

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

**Làm sao để ngăn chặn XSS trong Jquery?**

Loại XSS phổ biến nhất của jQuery là khi bạn gửi đầu cào trong jQuery selector. Những nhà phát triển web thường dùng location.hash và gửi nó đến selector gây ra XSS khi jQuery kết xuất HTML. jQuery đã nhận ra vấn đề này và vá lỗi logic của selector bằng cách kiểm tra đầu vào bắt đầu với hash. Bây giờ jQuery sẽ chỉ kết xuất HTML nếu kí tự đầu tiên là <. Nếu bạn gửi dữ liệu không đáng tin cậy đến selector, hãy đảm bảo bạn escape đúng các giá trị bằng cách dùng hàm jsEscape trên.

**Giảm thiểu XSS bằng cách dùng CSP**

CSP là cách phòng thủ cuối chống lại XSS. Nếu việc ngăn chặn XSS của bạn thất bại, bạn có thể dùng CSP để giảm thiểu lỗi XSS bằng cách hạn chế những việc kẻ tấn công có thể làm.

CSP cho bạn kiểm soát rất nhiều thứ, những tập lệnh bên ngoài có thể được tải hay tập lệnh inline sẽ được thực thi không. Để triển khai CSP bạn cần header Content-Security-Policy trong phản hồi với giá trị của nó

Ví dụ:

default-src ‘self’; script-src ‘self’; object-src ‘none’; frame-src ‘none’; base-uri ‘none’;

Chính sách này chỉ định những mã nguồn như hình ảnh hay tập lệnh chỉ có thể được tải từ chính trang web đó. Vì vậy, thậm chí nếu kẻ tấn công có thể tiêm XSS thành công cũng chỉ có thể tải những mã nguồn từ trang đó thôi. Điều này giảm cơ hội kẻ tẫn công có thể khai thác lỗ hổng XSS.

Nếu bạn yêu cầu tải những mã nguồn từ bên ngoài, đảm bảo rằng bạn chỉ cho phép tải những tập lệnh sẽ không hỗ trợ kẻ tấn công khai thác trang web này. Ví dụ, nếu bạn whitelist những tên miền cố đinh, sau đó kẻ tấn công có thể tải bất cứ mã nguồn nào từ tên miền đó. Nếu có thể , hãy lưu trữ những mã nguồn đó trên chính tên miền của bạn.

Nếu không thể, sau đó bạn có thể dùng hash- hay chính sách nonce-based cho phép tải những tập lệnh trên các tên miền khác nhau. Nonce là 1 chuỗi ngẫu nhiên được thêm như 1 thuộc tính của tập lệnh hay mã nguồn, sẽ chỉ thực thi nếu chuỗi ngẫu nhiên đó giống với cái máy chủ tạo ra. Một kẻ tấn công không thể đoán chuỗi đó và do đó không thể gọi kịch bản hay mã nguồn với 1 giá trị nonce không hợp lệ nên mã nguồn sẽ không được thực thi.