**CSRF**

Trong phần này, chúng tôi sẽ giải thích yêu cầu giả mạo chéo trang là gì, mô tả một số ví dụ về lỗ hổng CSRF phổ biến và giải thích cách ngăn chặn các cuộc tấn công CSRF.

**BÀI 1: CSRF LÀ GÌ ? (\*)**

Giả mạo yêu cầu chéo trang (còn được gọi là CSRF – Cross-site request forgery) là một **lỗ hổng bảo mật** web **cho phép kẻ tấn công dụ dỗ người dùng thực hiện các hành động mà họ không có ý định thực hiện**. Nó cho phép kẻ tấn công **tránh một phần chính sách cùng nguồn gốc, được thiết kế để ngăn các trang web khác nhau can thiệp lẫn nhau.**

**TÁC ĐỘNG CỦA TẤN CÔNG CSRF LÀ GÌ?**

Trong một cuộc tấn công CSRF thành công, kẻ tấn công **khiến người dùng nạn nhân thực hiện một hành động vô ý.** Ví dụ, có thể là **thay đổi địa chỉ email trên tài khoản của họ, thay đổi mật khẩu hoặc thực hiện chuyển tiền**. **Tùy thuộc vào bản chất của hành động, kẻ tấn công có thể giành được quyền kiểm soát hoàn toàn đối với tài khoản của người dùng**. Nếu người dùng bị xâm phạm có vai trò đặc quyền trong ứng dụng, thì kẻ tấn công **có thể giành được quyền kiểm soát hoàn toàn đối với tất cả dữ liệu và chức năng của ứng dụng.**

**CSRF HOẠT ĐỘNG THẾ NÀO?**

Để có thể thực hiện tấn công CSRF, phải **có 3 điều kiện chính**:

* Một hành động có liên quan. **Có một hành động trong ứng dụng mà kẻ tấn công có lý do để thực hiện. Đây có thể là một hành động đặc quyền** (chẳng hạn như sửa đổi quyền cho người dùng khác) **hoặc bất kỳ hành động nào trên dữ liệu cụ thể của người dùng** (chẳng hạn như thay đổi mật khẩu của chính người dùng).
* **Xử lý phiên dựa trên cookie**. **Thực hiện hành động liên quan đến** việc **phát hành một hoặc nhiều yêu cầu HTTP**, **và ứng dụng chỉ dựa vào cookie phiên để xác định người dùng đã thực hiện các yêu cầu.** **Không có cơ chế nào khác để theo dõi các phiên hoặc xác thực các yêu cầu của người dùng.**
* **Không có tham số yêu cầu không thể đoán trước.** Các **request thực hiện hành động không chứa bất kỳ tham số nào** mà **kẻ tấn công không thể xác định hoặc đoán giá trị của chúng**. **Ví dụ: khi khiến người dùng thay đổi mật khẩu của họ, chức năng này không dễ bị tấn công nếu kẻ tấn công cần biết giá trị của mật khẩu hiện tại.**

Ví dụ, giả sử một ứng dụng có một chức năng cho phép người dùng thay đổi địa chỉ email trên tài khoản của họ. Khi người dùng thực hiện hành động này, họ sẽ thực hiện một yêu cầu HTTP như sau:

POST /email/change HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 30

Cookie: session=yvthwsztyeQkAPzeQ5gHgTvlyxHfsAfE

[email=wiener@normal-user.com](mailto:email=wiener@normal-user.com)

Điều này đáp ứng các điều kiện bắt buộc đối với CSRF:

* **Hành động thay đổi địa chỉ email trên tài khoản của người dùng là mối quan tâm của kẻ tấn công.** Sau hành động này, kẻ tấn công thường có thể kích hoạt đặt lại mật khẩu và kiểm soát hoàn toàn tài khoản của người dùng.
* **Ứng dụng sử dụng cookie phiên để xác định người dùng nào đã đưa ra yêu cầu**. Không có mã thông báo hoặc cơ chế nào khác để theo dõi phiên của người dùng.
* Kẻ tấn công có thể **dễ dàng xác định các giá trị của tham số yêu cầu cần thiết để thực hiện hành động.**

Với những điều kiện này, kẻ tấn công có thể xây dựng một trang web có chứa mã HTML sau:

<html>

<body>

<form action="https://vulnerable-website.com/email/change" method="POST">

<input type="hidden" name="email" value="pwned@evil-user.net" />

</form>

<script>

document.forms[0].submit();

</script>

</body>

</html>

**Nếu người dùng nạn nhân truy cập trang web của kẻ tấn công, những điều sau đây sẽ xảy ra**:

* **Trang của kẻ tấn công sẽ kích hoạt yêu cầu HTTP đến trang web dễ bị tấn công**.
* **Nếu người dùng đã đăng nhập vào trang web dễ bị tấn công**, **trình duyệt** của **họ sẽ tự động bao gồm cookie phiên của họ trong yêu cầu** (giả sử cookie SameSite không được sử dụng).
* **Trang web dễ bị tấn công sẽ xử lý yêu cầu theo cách thông thường, coi như yêu cầu đó được thực hiện bởi người dùng nạn nhân và thay đổi địa chỉ email của họ.**

Lưu ý:

Mặc **dù CSRF thường được mô tả liên quan đến việc xử lý phiên dựa trên cookie**, **nhưng** nó **cũng phát sinh trong các bối cảnh khác khi ứng dụng tự động thêm một số thông tin xác thực của người dùng vào các yêu cầu,** chẳng hạn **như xác thực HTTP cơ bản và xác thực dựa trên chứng chỉ.**

**CÁCH XÂY DỰNG MỘT CUỘC TẤN CÔNG CSRF**

Việc **tạo thủ công HTML** cần thiết cho một khai thác CSRF **có thể rất cồng kềnh**, **đặc biệt là khi yêu cầu mong muốn chứa một số lượng lớn các tham số hoặc có những điểm kỳ quặc khác trong yêu cầu**. Cách **dễ nhất** để xây dựng một khai thác CSRF **là sử dụng trình tạo CSRF PoC được tích hợp sẵn trong Burp Suite Professional:**

**CÁCH THỰC HIỆN KHAI THÁC CSRF**

**Cơ chế phân phối cho các cuộc tấn công** giả mạo yêu cầu chéo trang về c**ơ bản giống như đối với XSS phản ánh**. Thông thường, **kẻ tấn công sẽ đặt HTML độc hại vào một trang web mà chúng kiểm soát, sau đó dụ nạn nhân truy cập trang web đó**. Điều này **có thể được thực hiện bằng cách cung cấp cho người dùng một liên kết đến trang web, thông qua email hoặc tin nhắn trên mạng xã hội.** Hoặc **nếu cuộc tấn công được thực hiện vào một trang web phổ biến** (ví dụ: trong bình luận của người dùng), **chúng có thể chỉ cần đợi người dùng truy cập trang web.**

Lưu ý rằng **một số khai thác CSRF đơn giản sử dụng phương thức GET và có thể hoàn toàn độc lập với một URL duy nhất trên trang web dễ bị tấn công**. Trong tình huống này, **kẻ tấn công có thể không cần sử dụng một trang web bên ngoài và có thể trực tiếp cung cấp cho nạn nhân một URL độc hại trên miền dễ bị tấn công**. Trong ví dụ trước, n**ếu yêu cầu thay đổi địa chỉ email có thể được thực hiện bằng phương thức GET, thì một cuộc tấn công độc lập sẽ trông như thế này:**

<img src="https://vulnerable-website.com/email/change?email=pwned@evil-user.net">

**PHÒNG THỦ THÔNG THƯỜNG CHỐNG LẠI CSRF**

Ngày nay, việc tìm và khai thác thành công các lỗ hổng CSRF thường liên quan đến việc bỏ qua các biện pháp chống CSRF được triển khai bởi trang web mục tiêu, trình duyệt của nạn nhân hoặc cả hai. Các biện pháp phòng thủ phổ biến nhất mà bạn sẽ gặp phải như sau:

* **Mã thông báo CSRF** - Mã thông báo CSRF **là giá trị duy nhất, bí mật và không thể đoán trước được tạo bởi ứng dụng phía máy chủ và được chia sẻ với máy khách**. **Khi cố gắng thực hiện hành động nhạy cảm**, chẳng hạn như gửi biểu mẫu, **máy khách phải bao gồm mã thông báo CSRF chính xác trong yêu cầu. Điều này khiến kẻ tấn công rất khó xây dựng yêu cầu hợp lệ thay mặt cho nạn nhân.**
* **Cookie SameSite** - **SameSite là cơ chế bảo mật của trình duyệt xác định thời điểm cookie của trang web được bao gồm trong các yêu cầu bắt nguồn từ các trang web khác**. Vì **các yêu cầu thực hiện hành động nhạy cảm thường yêu cầu cookie phiên đã xác thực,** **nên các hạn chế SameSite phù hợp có thể ngăn kẻ tấn công kích hoạt các hành động này trên nhiều trang web.** Kể từ năm 2021, Chrome áp dụng các hạn chế SameSite lỏng lẻo theo mặc định. Vì đây là tiêu chuẩn được đề xuất, chúng tôi hy vọng các trình duyệt chính khác sẽ áp dụng hành vi này trong tương lai.
* **Xác thực dựa trên tham chiếu** - **Một số ứng dụng sử dụng tiêu đề HTTP Referer để cố gắng chống lại các cuộc tấn công CSRF**, thông **thường bằng cách xác minh rằng yêu cầu bắt nguồn từ miền riêng của ứng dụng. Nhìn chung, điều này kém hiệu quả hơn xác thực mã thông báo CSRF.**

Để biết mô tả chi tiết hơn về từng biện pháp phòng thủ này, cũng như cách chúng có thể bị bỏ qua, hãy xem các tài liệu sau. Chúng bao gồm các phòng thí nghiệm tương tác cho phép bạn thực hành những gì đã học trên các mục tiêu dễ bị tấn công thực tế và cố ý.

**BÀI 2: XSS VÀ CSRF (\*)**

Trong phần này, chúng tôi sẽ giải thích sự khác biệt giữa XSS và CSRF và thảo luận liệu mã thông báo CSRF có thể giúp ngăn chặn các cuộc tấn công XSS hay không.

**SỰ KHÁC BIỆT GIỮA XSS VÀ CSRF**

**Cross-site scripting (hay XSS) cho phép kẻ tấn công thực thi JavaScript tùy ý trong trình duyệt của người dùng nạn nhân.**

**Cross-site request forgery (hay CSRF) cho phép kẻ tấn công dụ dỗ người dùng nạn nhân thực hiện các hành động mà họ không có ý định thực hiện.**

Hậu quả của **lỗ hổng XSS thường nghiêm trọng hơn so với lỗ hổng CSRF**:

* **CSRF** **thường** chỉ **áp dụng cho một tập hợp con các hành động mà người dùng có thể thực hiện.** **Nhiều ứng dụng triển khai các biện pháp phòng thủ CSRF nói chung nhưng bỏ qua một hoặc hai hành động bị lộ**. Ngược lại, **một khai thác XSS thành công thường có thể dụ dỗ người dùng thực hiện bất kỳ hành động nào mà người dùng có thể thực hiện, bất kể chức năng nào mà lỗ hổng phát sinh.**
* **CSRF** có thể **được mô tả là lỗ hổng "một chiều",** trong khi **kẻ tấn công có thể dụ dỗ nạn nhân đưa ra yêu cầu HTTP**, họ **không thể lấy phản hồi từ yêu cầu đó**. Ngược lại, **XSS là "hai chiều",** trong đó **tập lệnh được tiêm của kẻ tấn công có thể đưa ra các yêu cầu tùy ý, đọc phản hồi và trích xuất dữ liệu vào miền bên ngoài mà kẻ tấn công lựa chọn.**

**MÃ CSRF CÓ THỂ NGĂN CHẶN XSS KHÔNG ?**

Một số cuộc tấn công XSS thực sự **có thể** được **ngăn chặn thông qua việc sử dụng hiệu quả các mã thông báo CSRF**. Hãy xem xét một lỗ hổng XSS phản ánh đơn giản có thể được khai thác một cách tầm thường như thế này:

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

Bây giờ, giả sử rằng hàm dễ bị tấn công bao gồm mã thông báo CSRF:

[https://insecure-website.com/status?csrf-token=CIwNZNlR4XbisJF39I8yWnWX9wX4WFoz&message=<script>/\*+Bad+stuff+here...+\*/</script](https://insecure-website.com/status?csrf-token=CIwNZNlR4XbisJF39I8yWnWX9wX4WFoz&message=%3cscript%3e/*+Bad+stuff+here...+*/%3c/script)>

Giả sử **máy chủ xác thực đúng mã thông báo CSRF và từ chối các yêu cầu không có mã thông báo hợp lệ, thì mã thông báo sẽ ngăn chặn việc khai thác lỗ hổng XSS**. Manh mối ở đây nằm ở tên gọi: "**cross-site scripting**", **ít nhất là ở dạng phản ánh** của nó, **liên quan đến một yêu cầu cross-site.** Bằng cách **ngăn chặn kẻ tấn công tạo ra một yêu cầu cross-site, ứng dụng sẽ ngăn chặn việc khai thác lỗ hổng XSS một cách tầm thường.**

Một số cảnh báo quan trọng phát sinh ở đây:

* **Nếu** lỗ hổng **XSS phản ánh tồn tại ở bất kỳ nơi nào khác trên trang web** trong một chức năng **không được bảo vệ bởi mã thông báo CSRF**, thì XSS đó **có thể bị khai thác theo cách thông thường.**
* **Nếu** lỗ hổng **XSS có thể khai thác tồn tại ở bất kỳ nơi nào trên trang web**, thì lỗ hổng đó **có thể được tận dụng để khiến người dùng nạn nhân thực hiện các hành động ngay cả khi các hành động đó được bảo vệ bởi mã thông báo CSRF**. Trong tình huống này, **tập lệnh của kẻ tấn công có thể yêu cầu trang có liên quan lấy mã thông báo CSRF hợp lệ, sau đó sử dụng mã thông báo đó để thực hiện hành động được bảo vệ**.
* Mã thông báo **CSRF không bảo vệ chống lại các lỗ hổng XSS đã lưu trữ**. **Nếu một trang được bảo vệ bởi mã thông báo CSRF cũng là điểm đầu ra cho lỗ hổng XSS đã lưu trữ, thì lỗ hổng XSS đó có thể bị khai thác theo cách thông thường và tải trọng XSS sẽ thực thi khi người dùng truy cập trang.**

**BÀI 3: BỎ QUA XÁC THỰC MÃ THÔNG BÁO CSRF**

Trong phần này, chúng tôi sẽ giải thích mã thông báo CSRF là gì, cách chúng bảo vệ chống lại các cuộc tấn công CSRF và cách bạn có thể vượt qua các biện pháp phòng vệ này.

**MÃ THÔNG BÁO CSRF LÀ GÌ?**

Mã thông báo CSRF là **giá trị duy nhất, bí mật và không thể đoán trước được tạo ra bởi ứng dụng phía máy chủ và được chia sẻ với máy khách**. Khi đưa ra **yêu cầu thực hiện hành động nhạy cảm, chẳng hạn như gửi biểu mẫu, máy khách phải bao gồm mã thông báo CSRF chính xác. Nếu không, máy chủ sẽ từ chối thực hiện hành động được yêu cầu.**

Một cách phổ biến để chia sẻ mã thông báo CSRF với máy khách là bao gồm chúng dưới dạng tham số ẩn trong biểu mẫu HTML, ví dụ:

<form name="change-email-form" action="/my-account/change-email" method="POST">

<label>Email</label>

<input required type="email" name="email" value="example@normal-website.com">

<input required type="hidden" name="csrf" value="50FaWgdOhi9M9wyna8taR1k3ODOR8d6u">

<button class='button' type='submit'> Update email </button>

</form>

Việc gửi biểu mẫu này sẽ dẫn đến yêu cầu sau:

POST /my-account/change-email HTTP/1.1

Host: normal-website.com

Content-Length: 70

Content-Type: application/x-www-form-urlencoded

[csrf=50FaWgdOhi9M9wyna8taR1k3ODOR8d6u&email=example@normal-website.com](mailto:csrf=50FaWgdOhi9M9wyna8taR1k3ODOR8d6u&email=example@normal-website.com)

Khi được triển khai đúng cách, mã thông báo CSRF giúp **bảo vệ chống lại các cuộc tấn công CSRF bằng cách khiến kẻ tấn công khó có thể xây dựng một yêu cầu hợp lệ thay mặt cho nạn nhân.** **Vì kẻ tấn công không có cách nào dự đoán giá trị chính xác cho mã thông báo CSRF, nên chúng sẽ không thể đưa nó vào yêu cầu độc hại.**

Lưu ý

**Mã** thông báo **CSRF không cần phải được gửi dưới dạng tham số ẩn trong yêu cầu POST**. Ví dụ, **một số ứng dụng đặt mã thông báo CSRF trong tiêu đề HTTP**. **Cách mã thông báo được truyền có tác động đáng kể đến tính bảo mật của toàn bộ cơ chế. Để biết thêm thông tin, hãy xem Cách ngăn chặn lỗ hổng CSRF.**

**CÁC LỖI PHỔ BIẾN TRONG XÁC THỰC MÃ THÔNG BÁO CSRF**

Lỗ hổng CSRF **thường phát sinh do lỗi xác thực mã thông báo CSRF**. Trong phần này, chúng tôi sẽ đề cập đến một số vấn đề phổ biến nhất cho phép kẻ tấn công vượt qua các biện pháp phòng thủ này.

XÁC THỰC MÃ THÔNG BÁO CSRF **TÙY THUỘC VÀO PHƯƠNG THỨC YÊU CẦU**

Một số ứng dụng **xác thực** đúng **mã thông báo khi yêu cầu sử dụng phương thức POST nhưng bỏ qua xác thực khi sử dụng phương thức GET.**

Trong tình huống này, **kẻ tấn công có thể chuyển sang phương thức GET để bỏ qua xác thực và thực hiện tấn công CSRF:**

GET /email/change?email=pwned@evil-user.net HTTP/1.1

Host: vulnerable-website.com

Cookie: session=2yQIDcpia41WrATfjPqvm9tOkDvkMvLm

VIỆC **XÁC THỰC MÃ THÔNG BÁO CSRF PHỤ THUỘC VÀO MÃ THÔNG BÁO HIỆN CÓ**

Một số ứng dụng **xác thực đúng mã thông báo khi nó có mặt nhưng bỏ qua xác thực nếu mã thông báo bị bỏ qua.**

Trong tình huống này, kẻ tấn công **có thể xóa toàn bộ tham số chứa mã thông báo (không chỉ giá trị của nó) để bỏ qua xác thực và thực hiện tấn công CSRF:**

POST /email/change HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 25

Cookie: session=2yQIDcpia41WrATfjPqvm9tOkDvkMvLm

[email=pwned@evil-user.net](mailto:email=pwned@evil-user.net)

**MÃ THÔNG BÁO** CSRF **KHÔNG BỊ RÀNG BUỘC VỚI PHIÊN NGƯỜI DÙNG**

Một số **ứng dụng không xác thực rằng mã thông báo thuộc về cùng một phiên với người dùng đang thực hiện yêu cầu.** Thay vào đó, **ứng dụng duy trì một nhóm mã thông báo toàn cầu mà nó đã phát hành và chấp nhận bất kỳ mã thông báo nào xuất hiện trong nhóm này**.

Trong tình huống này, **kẻ tấn công có thể đăng nhập vào ứng dụng bằng tài khoản của riêng họ, lấy mã thông báo hợp lệ, sau đó cung cấp mã thông báo đó cho người dùng nạn nhân trong cuộc tấn công CSRF của họ.**

MÃ THÔNG BÁO **CSRF ĐƯỢC GẮN VỚI COOKIE KHÔNG CÓ PHIÊN**

Trong một biến thể của lỗ hổng trước đó, một số **ứng dụng liên kết mã thông báo CSRF với cookie, nhưng không phải với cùng một cookie được sử dụng để theo dõi phiên**. Điều này **có thể dễ dàng xảy ra khi một ứng dụng sử dụng hai khuôn khổ khác nhau, một để xử lý phiên và một để bảo vệ CSRF, không được tích hợp với nhau:**

POST /email/change HTTP/1.1

Host: vulnerability-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 68

Cookie: session=pSJYSScWKpmC60LpFOAHKixuFuM4uXWF; csrfKey=rZHCnSzEp8dbI6atzagGoSYyqJqTz5dv

[csrf=RhV7yQDO0xcq9gLEah2WVbmuFqyOq7tY&email=wiener@normal-user.com](mailto:csrf=RhV7yQDO0xcq9gLEah2WVbmuFqyOq7tY&email=wiener@normal-user.com)

**Tình huống này khó khai thác hơn nhưng vẫn dễ bị tấn công**. **Nếu** trang **web có bất kỳ hành vi nào cho phép kẻ tấn công đặt cookie trong trình duyệt của nạn nhân**, thì c**ó thể xảy ra tấn công.** **Kẻ tấn công có thể đăng nhập vào ứng dụng bằng tài khoản của riêng họ, lấy mã thông báo hợp lệ và cookie liên quan, tận dụng hành vi đặt cookie để đặt cookie của họ vào trình duyệt của nạn nhân và cung cấp mã thông báo của họ cho nạn nhân trong cuộc tấn công CSRF của họ.**

Lưu ý

**Hành vi thiết lập cookie thậm chí không cần phải tồn tại trong cùng một ứng dụng web như lỗ hổng CSRF.** **Bất kỳ ứng dụng nào** khác **trong cùng một miền DNS tổng thể đều có khả năng được tận dụng để thiết lập cookie trong ứng dụng đang bị nhắm mục tiêu**, **nếu cookie được kiểm soát có phạm vi phù hợp**. Ví dụ: **chức năng thiết lập cookie trên staging.demo.normal-website.com có ​​thể được tận dụng để đặt cookie được gửi đến secure.normal-website.com.**

**MÃ** THÔNG BÁO **CSRF ĐƯỢC SAO CHÉP ĐƠN GIẢN TRONG COOKIE**

Trong một biến thể khác của lỗ hổng trước đó, một số **ứng dụng không duy trì bất kỳ bản ghi nào về phía máy chủ của các mã thông báo đã được phát hành**, mà **thay vào đó sao chép từng mã thông báo trong cookie và tham số yêu cầu**. **Khi yêu cầu tiếp theo được xác thực**, **ứng dụng chỉ cần xác minh rằng mã thông báo được gửi trong tham số yêu cầu khớp với giá trị được gửi trong cookie.** Điều này đôi khi được g**ọi là biện pháp phòng thủ "gửi kép" chống lại CSRF và được ủng hộ vì nó dễ triển khai và tránh nhu cầu về bất kỳ trạng thái nào ở phía máy chủ:**

POST /email/change HTTP/1.1

Host: vulnerability-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 68

Cookie: session=1DQGdzYbOJQzLP7460tfyiv3do7MjyPw; csrf=R8ov2YBfTYmzFyjit8o2hKBuoIjXXVpa

[csrf=R8ov2YBfTYmzFyjit8o2hKBuoIjXXVpa&email=wiener@normal-user.com](mailto:csrf=R8ov2YBfTYmzFyjit8o2hKBuoIjXXVpa&email=wiener@normal-user.com)

Trong tình huống này, **kẻ tấn công có thể thực hiện lại một cuộc tấn công CSRF nếu trang web có bất kỳ chức năng cài đặt cookie nào.** Ở đây, **kẻ tấn công không cần phải có được một mã thông báo hợp lệ của riêng họ.** Họ **chỉ cần tạo ra một mã thông báo** (có thể ở định dạng bắt buộc, nếu định dạng đó đang được kiểm tra), **tận dụng hành vi cài đặt cookie để đặt cookie của họ vào trình duyệt của nạn nhân và cung cấp mã thông báo của họ cho nạn nhân trong cuộc tấn công CSRF của họ.**

**BÀI 4: BỎ QUA HẠN CHẾ VỀ COOKIE SAMESITE**

**SameSite** **là cơ chế bảo mật của trình duyệt xác định thời điểm cookie của trang web được đưa vào các yêu cầu bắt nguồn từ các trang web khác**. Các hạn chế cookie của SameSite cung cấp khả năng bảo vệ một phần chống lại nhiều cuộc tấn công xuyên trang, bao gồm CSRF, rò rỉ xuyên trang và một số khai thác CORS.

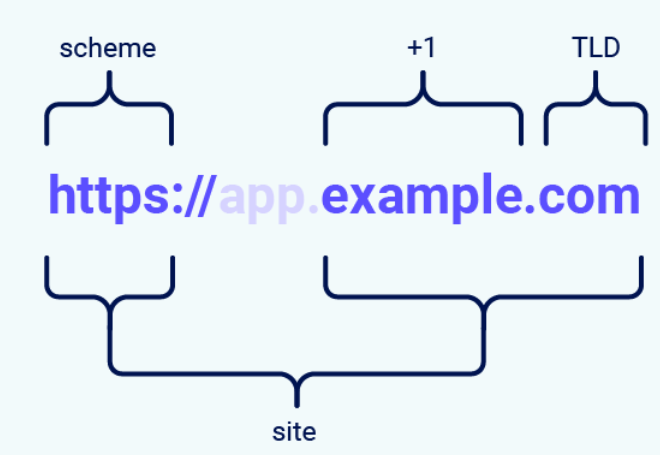
Từ năm 2021, Chrome áp dụng các hạn chế SameSite lỏng lẻo theo mặc định nếu trang web phát hành cookie không đặt rõ mức hạn chế của riêng mình. Đây là một tiêu chuẩn được đề xuất và chúng tôi hy vọng các trình duyệt chính khác sẽ áp dụng hành vi này trong tương lai. Do đó, điều cần thiết là phải nắm vững cách thức hoạt động của các hạn chế này cũng như cách chúng có khả năng bị bỏ qua để kiểm tra kỹ lưỡng các vectơ tấn công xuyên trang.

Trong phần này, trước tiên chúng ta sẽ đề cập đến cách thức hoạt động của cơ chế SameSite và làm rõ một số thuật ngữ liên quan. Sau đó, chúng ta sẽ xem xét một số cách phổ biến nhất mà bạn có thể bỏ qua các hạn chế này, cho phép CSRF và các cuộc tấn công xuyên trang khác trên các trang web ban đầu có vẻ an toàn.

\* **MỘT TRANG WEB TRONG BỐI CẢNH COOKIE SAMESITE LÀ GÌ** ?

Trong bối cảnh hạn chế cookie SameSite, **một trang web được định nghĩa là tên miền cấp cao nhất (TLD), thường là một cái gì đó như .com hoặc .net, cộng với một cấp bổ sung của tên miền. Điều này thường được gọi là TLD+1.**

Khi **xác định** xem **một yêu cầu có phải là cùng một trang web hay không**, **lược đồ (scheme) URL cũng được xem xét.** Điều này có **nghĩa là một liên kết từ http://app.example.com đến https://app.example.com được coi là liên kết chéo trang web bởi hầu hết các trình duyệt.**



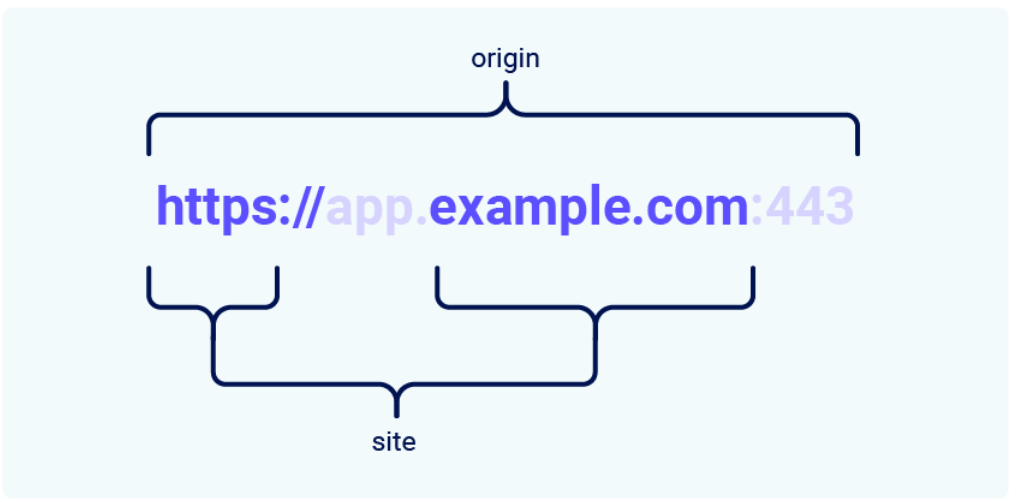
Lưu ý

Bạn **có thể bắt gặp thuật ngữ "tên miền cấp cao hiệu quả" (eTLD).** **Đây chỉ là một cách để giải thích cho các hậu tố nhiều phần được bảo lưu được coi là tên miền cấp cao trong thực tế, chẳng hạn như .co.uk.**

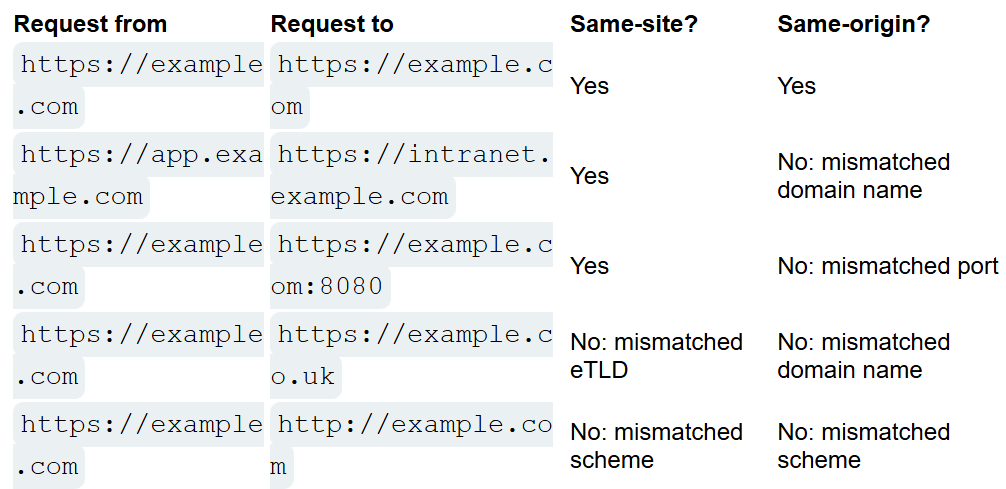
\* **SỰ KHÁC BIỆT GIỮA MỘT TRANG WEB VÀ 1 NGUỒN GỐC LÀ GÌ ?**

Sự **khác biệt giữa site và origin** **là phạm vi của chúng; site bao gồm nhiều tên miền, trong khi origin chỉ bao gồm một.** Mặc **dù chúng có liên quan chặt chẽ, nhưng điều quan trọng là không sử dụng các thuật ngữ này thay thế cho nhau** **vì việc gộp chung hai thuật ngữ này có thể gây ra những tác động nghiêm trọng đến bảo mật.**

**Hai URL được coi là có cùng nguồn gốc nếu chúng chia sẻ cùng một lược đồ, tên miền và cổng chính xác**. Mặc dù lưu ý rằng cổng thường được suy ra từ lược đồ.



Như bạn có thể thấy từ ví dụ này, **thuật ngữ "site" ít cụ thể hơn nhiều vì nó chỉ tính đến lược đồ và phần cuối của tên miền. Quan trọng là, điều này có nghĩa là yêu cầu cross-origin vẫn có thể là cùng một site, nhưng không phải ngược lại.**



**Đây là** một sự **khác biệt quan trọng vì** nó **có nghĩa là bất kỳ lỗ hổng nào cho phép thực thi JavaScript tùy ý đều có thể bị lạm dụng để vượt qua các biện pháp phòng thủ dựa trên trang web trên các tên miền khác thuộc cùng một trang web. Chúng ta sẽ xem một ví dụ về điều này trong một trong các phòng thí nghiệm sau.**

\* **SAME SITE HOẠT ĐỘNG NHƯ THẾ NÀO**?

**Trước khi cơ chế SameSite được giới thiệu**, **trình duyệt** đã **gửi cookie trong mọi yêu cầu đến miền đã phát hành chún**g, ngay cả khi yêu cầu được kích hoạt bởi một trang web của bên thứ ba không liên quan. SameSite hoạt động bằng cách cho phép trình duyệt và chủ sở hữu trang web giới hạn các yêu cầu liên trang web nào, nếu có, nên bao gồm các cookie cụ thể. Điều này có thể giúp giảm khả năng người dùng tiếp xúc với các cuộc tấn công CSRF, khiến trình duyệt của nạn nhân phát hành một yêu cầu kích hoạt hành động có hại trên trang web dễ bị tấn công. Vì các yêu cầu này thường yêu cầu cookie được liên kết với phiên đã xác thực của nạn nhân, nên cuộc tấn công sẽ thất bại nếu trình duyệt không bao gồm cookie này.

Tất cả các trình duyệt chính hiện hỗ trợ các mức hạn chế SameSite sau:

* Strict
* Lax
* None

Các nhà phát triển có thể cấu hình thủ công mức hạn chế cho mỗi cookie mà họ đặt, giúp họ kiểm soát tốt hơn thời điểm sử dụng các cookie này. Để thực hiện việc này, họ chỉ cần đưa thuộc tính SameSite vào tiêu đề phản hồi Set-Cookie cùng với giá trị ưa thích của họ:

Set-Cookie: session=0F8tgdOhi9ynR1M9wa3ODa; SameSite=Strict

Mặc dù điều này cung cấp một số biện pháp bảo vệ chống lại các cuộc tấn công CSRF, nhưng không có hạn chế nào trong số này cung cấp khả năng miễn dịch được đảm bảo, vì chúng tôi sẽ chứng minh bằng cách sử dụng các phòng thí nghiệm tương tác dễ bị tấn công một cách có chủ đích sau trong phần này

Lưu ý

Nếu trang web phát hành cookie không thiết lập rõ ràng thuộc tính SameSite, Chrome sẽ tự động áp dụng các hạn chế Lax theo mặc định. Điều này có nghĩa là cookie chỉ được gửi trong các yêu cầu liên trang web đáp ứng các tiêu chí cụ thể, ngay cả khi các nhà phát triển không bao giờ cấu hình hành vi này. Vì đây là một tiêu chuẩn mới được đề xuất, chúng tôi hy vọng các trình duyệt chính khác sẽ áp dụng hành vi này trong tương lai.

\* STRICT

Nếu cookie được đặt với thuộc tính SameSite=Strict, trình duyệt sẽ không gửi cookie đó trong bất kỳ yêu cầu liên trang web nào. Nói một cách đơn giản, điều này có nghĩa là nếu trang web mục tiêu cho yêu cầu không khớp với trang web hiện đang hiển thị trên thanh địa chỉ của trình duyệt, thì cookie đó sẽ không bao gồm.

Điều này được khuyến nghị khi đặt cookie cho phép người mang sửa đổi dữ liệu hoặc thực hiện các hành động nhạy cảm khác, chẳng hạn như truy cập các trang cụ thể chỉ dành cho người dùng đã xác thực.

Mặc dù đây là tùy chọn an toàn nhất, nhưng nó có thể ảnh hưởng tiêu cực đến trải nghiệm của người dùng trong trường hợp chức năng liên trang web là mong muốn.

\* LAX:

Các hạn chế của SameSite lỏng lẻo có nghĩa là trình duyệt sẽ gửi cookie trong các yêu cầu liên trang web, nhưng chỉ khi cả hai điều kiện sau được đáp ứng:

* Yêu cầu sử dụng phương thức GET.
* Yêu cầu xuất phát từ điều hướng cấp cao nhất của người dùng, chẳng hạn như nhấp vào liên kết.

Điều này có nghĩa là cookie không được bao gồm trong các yêu cầu POST liên trang web, chẳng hạn. Vì các yêu cầu POST thường được sử dụng để thực hiện các hành động sửa đổi dữ liệu hoặc trạng thái (ít nhất là theo thông lệ tốt nhất), nên chúng có nhiều khả năng trở thành mục tiêu của các cuộc tấn công CSRF hơn.

Tương tự như vậy, cookie không được bao gồm trong các yêu cầu nền, chẳng hạn như các yêu cầu được khởi tạo bởi tập lệnh, khung nội tuyến hoặc tham chiếu đến hình ảnh và các tài nguyên khác.

\* NONE:

Nếu cookie được đặt với thuộc tính SameSite=None, điều này sẽ vô hiệu hóa hoàn toàn các hạn chế của SameSite, bất kể trình duyệt nào. Do đó, trình duyệt sẽ gửi cookie này trong mọi yêu cầu đến trang web đã phát hành cookie, ngay cả những yêu cầu được kích hoạt bởi các trang web của bên thứ ba hoàn toàn không liên quan.

Ngoại trừ Chrome, đây là hành vi mặc định được các trình duyệt chính sử dụng nếu không cung cấp thuộc tính SameSite khi đặt cookie.

Có những lý do chính đáng để vô hiệu hóa SameSite, chẳng hạn như khi cookie được dự định sử dụng từ ngữ cảnh của bên thứ ba và không cấp cho người mang quyền truy cập vào bất kỳ dữ liệu hoặc chức năng nhạy cảm nào. Cookie theo dõi là một ví dụ điển hình.

Nếu bạn gặp phải cookie được đặt với SameSite=None hoặc không có hạn chế rõ ràng, bạn nên điều tra xem nó có hữu ích không. Khi hành vi "Lax-by-default" lần đầu tiên được Chrome áp dụng, điều này có tác dụng phụ là phá vỡ nhiều chức năng web hiện có. Để giải quyết nhanh chóng, một số trang web đã chọn chỉ cần vô hiệu hóa các hạn chế của SameSite đối với tất cả cookie, bao gồm cả những cookie có khả năng nhạy cảm.

Khi thiết lập cookie với SameSite=None, trang web cũng phải bao gồm thuộc tính Secure, đảm bảo rằng cookie chỉ được gửi trong các tin nhắn được mã hóa qua HTTPS. Nếu không, trình duyệt sẽ từ chối cookie và cookie sẽ không được thiết lập.

Set-Cookie: trackingId=0F8tgdOhi9ynR1M9wa3ODa; SameSite=None; Secure

\* VƯỢT QUA NHỮNG HẠN CHẾ CỦA SAMESITE LAX BẰNG SỬ DỤNG GET REQUESTS

Trên thực tế, máy chủ không phải lúc nào cũng cầu kỳ về việc chúng có nhận được yêu cầu GET hay POST đến một điểm cuối nhất định hay không, ngay cả những máy chủ đang mong đợi một biểu mẫu được gửi. Nếu chúng cũng sử dụng các hạn chế Lax cho cookie phiên của chúng, dù là rõ ràng hay do mặc định của trình duyệt, bạn vẫn có thể thực hiện một cuộc tấn công CSRF bằng cách tạo ra một yêu cầu GET từ trình duyệt của nạn nhân.

Miễn là yêu cầu liên quan đến điều hướng cấp cao nhất, trình duyệt vẫn sẽ bao gồm cookie phiên của nạn nhân. Sau đây là một trong những cách tiếp cận đơn giản nhất để khởi chạy một cuộc tấn công như vậy:

<script>

document.location = 'https://vulnerable-website.com/account/transfer-payment?recipient=hacker&amount=1000000';

</script>

Ngay cả khi yêu cầu GET thông thường không được phép, một số khung cung cấp cách ghi đè phương thức được chỉ định trong dòng yêu cầu. Ví dụ, Symfony hỗ trợ tham số \_method trong biểu mẫu, có quyền ưu tiên hơn phương thức thông thường cho mục đích định tuyến:

<form action="https://vulnerable-website.com/account/transfer-payment" method="POST">

<input type="hidden" name="\_method" value="GET">

<input type="hidden" name="recipient" value="hacker">

<input type="hidden" name="amount" value="1000000">

</form>

Các khuôn khổ khác hỗ trợ nhiều tham số tương tự.

\* VƯỢT QUA CÁC HẠN CHẾ CỦA SAMESITE BẰNG CÁCH SỬ DỤNG CÁC TIỆN ÍCH TẠI CHỖ

Nếu cookie được đặt với thuộc tính SameSite=Strict, trình duyệt sẽ không bao gồm cookie đó trong bất kỳ yêu cầu liên trang nào. Bạn có thể khắc phục hạn chế này nếu tìm được tiện ích tạo ra yêu cầu thứ cấp trong cùng một trang.

Một tiện ích khả thi là chuyển hướng phía máy khách, tạo động mục tiêu chuyển hướng bằng cách sử dụng đầu vào có thể kiểm soát được của kẻ tấn công như tham số URL. Để biết một số ví dụ, hãy xem tài liệu của chúng tôi về chuyển hướng mở dựa trên DOM.

Đối với trình duyệt, các chuyển hướng phía máy khách này không thực sự là chuyển hướng; yêu cầu kết quả chỉ được coi là yêu cầu độc lập thông thường. Quan trọng nhất, đây là yêu cầu cùng trang và do đó sẽ bao gồm tất cả cookie liên quan đến trang, bất kể bất kỳ hạn chế nào được áp dụng.

Nếu bạn có thể thao túng tiện ích này để tạo ra yêu cầu thứ cấp độc hại, điều này có thể cho phép bạn bỏ qua hoàn toàn mọi hạn chế cookie SameSite.

Lưu ý rằng không thể thực hiện cuộc tấn công tương đương với chuyển hướng phía máy chủ. Trong trường hợp này, trình duyệt nhận ra rằng yêu cầu theo dõi chuyển hướng ban đầu là kết quả của yêu cầu liên trang web, do đó chúng vẫn áp dụng các hạn chế cookie phù hợp.

\* VƯỢT QUA CÁC HẠN CHẾ CỦA SAMESITE QUA CÁC MIỀN ANH CHỊ EM DỄ BỊ TẤN TÔNG

Cho dù bạn đang thử nghiệm trang web của người khác hay cố gắng bảo mật trang web của riêng mình, điều quan trọng cần lưu ý là yêu cầu vẫn có thể là cùng một trang web ngay cả khi được phát hành từ nguồn gốc chéo.

Đảm bảo bạn kiểm tra kỹ lưỡng tất cả các bề mặt tấn công khả dụng, bao gồm bất kỳ tên miền anh chị em nào. Đặc biệt, các lỗ hổng cho phép bạn tạo ra một yêu cầu thứ cấp tùy ý, chẳng hạn như XSS, có thể xâm phạm hoàn toàn các biện pháp phòng thủ dựa trên trang web, khiến tất cả các tên miền của trang web bị tấn công chéo.

Ngoài CSRF cổ điển, đừng quên rằng nếu trang web mục tiêu hỗ trợ WebSockets, chức năng này có thể dễ bị tấn công WebSocket chéo trang web (CSWSH), về cơ bản chỉ là một cuộc tấn công CSRF nhắm vào bắt tay WebSocket. Để biết thêm chi tiết, hãy xem chủ đề của chúng tôi về các lỗ hổng WebSocket.

\* VƯỢT QUA CÁC HẠN CHẾ CỦA SAMESITE LAX BẰNG CÁC COOKIE MỚI ĐƯỢC PHÁT HÀNH

Cookie có hạn chế Lax SameSite thường không được gửi trong bất kỳ yêu cầu POST giữa các trang web nào, nhưng có một số trường hợp ngoại lệ.

Như đã đề cập trước đó, nếu một trang web không bao gồm thuộc tính SameSite khi đặt cookie, Chrome sẽ tự động áp dụng giới hạn Lax theo mặc định. Tuy nhiên, để tránh phá vỡ cơ chế đăng nhập một lần (SSO), Chrome không thực sự áp dụng các giới hạn này trong 120 giây đầu tiên đối với các yêu cầu POST cấp cao nhất. Do đó, có một cửa sổ hai phút mà người dùng có thể dễ bị tấn công chéo trang.

Lưu ý

Cửa sổ hai phút này không áp dụng cho các cookie được thiết lập rõ ràng bằng thuộc tính SameSite=Lax.

Sẽ không thực tế khi cố gắng căn thời gian cho cuộc tấn công rơi vào khoảng thời gian ngắn này. Mặt khác, nếu bạn có thể tìm thấy một tiện ích trên trang web cho phép bạn buộc nạn nhân phải cấp một cookie phiên mới, bạn có thể làm mới cookie của họ trước khi thực hiện cuộc tấn công chính. Ví dụ: hoàn tất luồng đăng nhập dựa trên OAuth có thể dẫn đến một phiên mới mỗi lần vì dịch vụ OAuth không nhất thiết phải biết liệu người dùng có vẫn đăng nhập vào trang web mục tiêu hay không.

Để kích hoạt làm mới cookie mà không cần nạn nhân phải đăng nhập lại theo cách thủ công, bạn cần sử dụng điều hướng cấp cao nhất, đảm bảo rằng các cookie được liên kết với phiên OAuth hiện tại của họ được bao gồm. Điều này đặt ra một thách thức bổ sung vì sau đó bạn cần chuyển hướng người dùng trở lại trang web của mình để bạn có thể khởi chạy cuộc tấn công CSRF.

Ngoài ra, bạn có thể kích hoạt làm mới cookie từ một tab mới để trình duyệt không rời khỏi trang trước khi bạn có thể thực hiện cuộc tấn công cuối cùng. Một trở ngại nhỏ với cách tiếp cận này là trình duyệt chặn các tab bật lên trừ khi chúng được mở thông qua tương tác thủ công. Ví dụ, theo mặc định, cửa sổ bật lên sau sẽ bị trình duyệt chặn:

window.open('https://vulnerable-website.com/login/sso');

Để giải quyết vấn đề này, bạn có thể gói câu lệnh trong trình xử lý sự kiện onclick như sau:

window.onclick = () => {

window.open('https://vulnerable-website.com/login/sso');

}

Theo cách này, phương thức window.open() chỉ được gọi khi người dùng nhấp vào đâu đó trên trang.

**BÀI 5: BỎ QUA CÁC BIỆN PHÁP PHÒNG THỦ CSRF DỰA TRÊN REFERER**

Ngoài các biện pháp phòng thủ sử dụng mã thông báo CSRF, một số ứng dụng sử dụng Referertiêu đề HTTP để cố gắng phòng thủ chống lại các cuộc tấn công CSRF, thường bằng cách xác minh rằng yêu cầu bắt nguồn từ miền riêng của ứng dụng. Cách tiếp cận này thường kém hiệu quả hơn và thường bị bỏ qua.

\* TIÊU ĐỀ THAM CHIẾU

Tiêu đề HTTP Referer (vô tình bị viết sai chính tả trong đặc tả HTTP) là tiêu đề yêu cầu tùy chọn có chứa URL của trang web liên kết đến tài nguyên đang được yêu cầu. Tiêu đề này thường được trình duyệt tự động thêm vào khi người dùng kích hoạt yêu cầu HTTP, bao gồm cả khi nhấp vào liên kết hoặc gửi biểu mẫu. Có nhiều phương pháp khác nhau cho phép trang liên kết giữ lại hoặc sửa đổi giá trị của tiêu Referer đề. Điều này thường được thực hiện vì lý do riêng tư.

\* XÁC THỰC CỦA REFERER PHỤ THUỘC VÀO VIỆC TIÊU ĐỀ CÓ HIỆN DIỆN HAY KHÔNG

Một số ứng dụng xác thực tiêu Refererđề khi tiêu đề có trong yêu cầu nhưng bỏ qua xác thực nếu tiêu đề bị bỏ sót.

Trong tình huống này, kẻ tấn công có thể tạo ra khai thác CSRF theo cách khiến trình duyệt của người dùng nạn nhân xóa Referertiêu đề trong yêu cầu kết quả. Có nhiều cách để thực hiện điều này, nhưng cách dễ nhất là sử dụng thẻ META trong trang HTML lưu trữ cuộc tấn công CSRF:

<meta name="referrer" content="never">

\* XÁC THỰC CỦA REFERRER CÓ THỂ BỊ BỎ QUA

Một số ứng dụng xác thực tiêu đề Referer theo cách ngây thơ có thể bị bỏ qua. Ví dụ: nếu ứng dụng xác thực rằng miền trong Referer bắt đầu bằng giá trị mong đợi, kẻ tấn công có thể đặt miền này làm miền phụ của miền riêng của chúng:

http://vulnerable-website.com.attacker-website.com/csrf-attack

Tương tự, nếu ứng dụng chỉ xác thực rằng Referer của nó chứa miền riêng của nó, kẻ tấn công có thể đặt giá trị bắt buộc ở nơi khác trong URL:

http://attacker-website.com/csrf-attack?vulnerable-website.com

Lưu ý

Mặc dù bạn có thể xác định hành vi này bằng Burp, nhưng bạn thường thấy rằng cách tiếp cận này không còn hiệu quả khi bạn kiểm tra bằng chứng khái niệm của mình trong trình duyệt. Để giảm nguy cơ dữ liệu nhạy cảm bị rò rỉ theo cách này, nhiều trình duyệt hiện xóa chuỗi truy vấn khỏi tiêu đề Referer theo mặc định.

Bạn có thể ghi đè hành vi này bằng cách đảm bảo rằng phản hồi chứa khai thác của bạn có tiêu đề Referrer-Policy: unsafe-url được đặt (lưu ý rằng tiêu đề Referrer được viết đúng chính tả trong trường hợp này, chỉ để đảm bảo bạn chú ý!). Điều này đảm bảo rằng URL đầy đủ được gửi đi, bao gồm cả chuỗi truy vấn.