**CSRF Protection và những điều cần phải biết**

4.739 lượt xem - Thời gian đọc: 5 phút đọc

Mọi người có tự hỏi vì sao một số email khi nhận được lại nằm trong thư mục spam?. Thành thật mà nói thì hãy thận trọng khi truy cập một trang web tràn ngập quảng cáo và chiêu dụ đá số đã được đính kèm trong mail đó.

Kẻ tấn công đằng sau đó có thể lừa bạn làm điều gì đó độc hại, chẳng hạn như xóa tài khoản của bạn trên một trang web, chuyển tiền bất hợp pháp, v.v. Đây là tất cả các kết quả có thể xảy ra từ một cuộc tấn công CSRF.

Các cuộc tấn công CSRF ngày nay không còn phổ biến bởi lẽ các công nghệ đã kiến trúc việc ngăn chặn nó từ bên trong lõi trước khi đến tay các lập trình viên xây dựng ứng dụng. Việc chúng ta hiểu cách chúng hoạt động là rất quan trọng nếu muốn xây dựng các dịch vụ và ứng dụng web an toàn.

**A Bird's-Eye View of CSRF**

CSRF là viết tắt cho **cross-site request forgery**

**Cross-Site Request**

**Cross-site request** đơn giản là gửi một yêu cầu từ trang B đến trang A. Nghe có vẻ khá là bình thường :)). Việc này chỉ bình thường khi chúng ta cho phép yêu cầu đó.

Cho ví dụ: Trong trường hợp mình muốn xoá tài khoản Firebase khỏi tài khoản Google của mình. Khá là bình thường đúng không?. Tuy nhiên, nếu mình cũng làm tương tự thế nhưng bằng cách sử dụng một tài khoản khác khong thuộc sở hữu Firebase, để xoá tài khoản Firebase khỏi tài khoản Google, thì đây rất có thể đã bị xâm hại bằng cách trên.

Đến đây có thể bạn bắt đầu đặt câu hỏi tại sao rồi đúng không? Tại sao mình muốn xóa tài khoản Firebase của mình bằng một số tài khoản, trang web ngẫu nhiên khác không có mối liên quan với nó?

Có thể có một vài trường hợp sử dụng phục vụ cho mục đích này. Ví dụ, mình có thể cho phép tài khoản GG Cloud được quyền xoá tài khoản Firebase của mình. Tương tự có thể cho phép tài khoản Facebook xoá tài khoản Instargram?. Tuy nhiên là, nếu bạn truy cập một trang web ngẫu nhiên nào đó khiến tất cả tài khoản Instagram hiện tại bị xoá sạch? bạn biết lý do rồi đấy

**Forgery (Hay còn gọi là giả mạo)**

Thuật ngữ "giả mạo" ở đây có nghĩa là thực hiện yêu cầu bất hợp pháp một hành động mà bạn không được phép làm thực hiện.

Vậy, CSRF hay còn gọi là **Cross-site Request Forgery** có nghĩa là mô tả một yêu cầu gửi đến phía máy chủ và nó không xác định được yêu cầu giả mạo được gửi đến. Vậ làm thế nào tin tặc có thể làm điều đó?

**Cuộc tấn công CSRF**

Nào nào, bây giờ hãy xem cách kẻ tấn công có thể thực hiện một cuộc tấn công CSRF vào ứng dụng của bạn.

**Ứng dụng cho cuộc thí nghiệm**

Giả sử ứng dụng của bạn có một trang chủ đơn giản và một trang profile. Trang chủ ứng dụng của bạn hiển thị cho bất kỳ ai trên web. Để ngắn gọn, ứng dụng sau đây hiển thị một trang đơn giản liệt kê một vài người dùng.

Tuy nhiên, để truy cập trang hồ sơ, người dùng phải được xác thực trên ứng dụng. Bên trong trang **profile**. sẽ cho một thực hiện một hành động là xoá tài khoản đang được xem.

**Luồng xác thực cho ứng dụng thử nghiệm**

Giả sử người dùng của bạn cố gắng đăng nhập vào ứng dụng của bạn bằng biểu mẫu đăng nhập. Người dùng điền vào biểu mẫu này để xác thực thông tin đăng nhập của họ từ máy chủ. Giống như hầu hết các luồng xác thực thông thường, máy chủ sẽ gửi một cookie được sử dụng để quản lý phiên của người dùng. Cookie này được lưu trữ trong trình duyệt và được gửi lại với mọi yêu cầu xác thực tính xác thực của người dùng.

**Lỗ hổng bảo mật**

Giả sử một người dùng muốn xóa tài khoản của họ trên trang web của bạn. Để làm điều này. Người dùng phải ấn vào nút **Xoá**. Tuy nhiên, chỉ người đã được nhập vài tài khoản mới được thực hiện điều này.

Khi người dùng ấn **Xoá**, phía máy khác sẽ gửi một yêu cầu đến máy chủ của bạn. Nó sẽ xử lý yêu cầu này và tính toán để xoá dữ liệu trên database. Yêu cầu xoá đó có dạng như thế này.

Để xác thực được yêu cầu **Xoá** này, trình duyệt của người dùng sẽ lưu trữ phiên token của máy chủ trong cookie. Tuy nhiên, điều này để lại lỗ hổng CSRF trong ứng dụng của bạn. Tin tặc có thể gửi yêu cầu **xoá** này với cookie đang được lưu trữ trên trình duyệt. Tất cả những gì chúng cần bạn làm là mở một liên kết có biểu mẫu ẩn kích hoạt yêu cầu xóa này trong nền ứng dụng. Hãy xem cách này hoạt động như thế nào.

**Tấn công**

Trong ví dụ này, điểm kích hoạt cuộc tấn công là mở một **Đường dẫn**. Kẻ tấn công tạo ra một URL trỏ đến một ứng dụng web khác. Sau đó, kẻ tấn công sử dụng kỹ thuật xã hội để mở URL đó trong trình duyệt của người dùng.

Ngay sau khi ứng dụng tải, nó có quyền truy cập vào cookie phiên được lưu trữ trong trình duyệt của bạn. !!!! Cuộc tấn công có thể được kích hoạt ẩn, trong nền, trong khi tải liên kết độc hại mà bận vừa ấn vào đấy. Và hậu quả thế nào thì bạn cũng biết rồi đấy. Đối tượng có thể sử dụng phiên token đã được lưu trữ ở trình duyệt để thực hiện bất cứ yêu cầu nào mà nó có thể.

**CSRF Protection: Myth Busters**

Để hiểu cách bạn có thể bảo vệ ứng dụng của mình khỏi cuộc tấn công **CSRF**, trước tiên bạn phải hiểu các giải pháp là không chắn chắn hoàn toàn. Những giải pháp này có vẻ dễ dàng, nhưng kẻ tấn công có thể dễ dàng qua mặt chúng. Và ứng dụng của bạn vẫn có thể dễ bị tấn công CSRF. Chúng ta hãy xem nhanh những điều này:

**Sử dụng Web Storage thay vì Cookies**

Bạn nghĩ răng có thể sử dụng các loại lưu trữ storage trên trình duyệt sẽ giải quyết được vấn đề này (localStorage, sessionStorage). Nhưng, kẻ tấn công lại có thể truy cập nó bất cứ lúc nào chỉ cần sử dụng

const token = localStorage.getItem("token");

**Sử dụng POST request**

Nếu bạn cấu trúc lại các điểm gọi đến máy chủ của mình với yêu cầu là phương thức **POST**. Bạn vẫn không hoàn toàn an toàn trước cuộc **tấn công CSRF**. Trong phần trước, mình đã minh họa một ví dụ về yêu cầu xóa để xóa tài khoản của người dùng. Đây cũng có thể là một yêu cầu **GET**, Tuy nhiên thì đổi sang **POST** cũng chẳng khác gì!

**CSRF Protection: The Reliable Solution**

**Sử dụng CORS**

Sử dụng **CORS** nói đơn giản là giao thức quyết định có cho phép một request lạ nào đó ở phía máy khách có thể được thực hiện hay không được vào nguồn gốc của nó (Domain).

// Dễ bị tấn công

app.get('/delete',(req,res)=>{

res.set('Access-Control-Allow-Origin', '\*');

...

})

// Chỉ cho phép các yêu cầu từ csrfprotection-client.com

app.get('/delete',(req,res)=>{

res.set('Access-Control-Allow-Origin', 'csrfprotection-client.com');

...

})

**Sử dụng CSRF Tokens**

CSRF Token, còn được gọi là Anti CSRF Token. Trước khi người dùng thực hiện gửi yêu cầu đến phía máy chủ, sẽ gửi yêu cầu lấy mã CSRF token để đính kèm cho yêu cầu sắp được gửi. Và tinh tế ở đây là token này chỉ tồn tại sau phiên đó, nó sẽ không được tái sử dụng cho các phiên yêu cầu khác. Tức là nó chỉ tồn tại trong một lần gửi yêu cầu.

Tuy nhiên, bạn cần đảm bảo rằng bạn không có bất kỳ lỗ hổng XSS nào trong ứng dụng của mình có thể làm rò rỉ các mã thông báo này cho kẻ tấn công. React đã bảo vệ bạn khỏi các cuộc tấn công XSS, nhưng đây là một hướng dẫn thú vị có thể giúp ích cho bạn nếu bạn muốn biết thêm.

Tham khảo [chi tiết](https://www.stackhawk.com/blog/react-csrf-protection-guide-examples-and-how-to-enable-it/)