**Cross-site request forgery (CSRF)**

**Khái niệm:**

CSRF là 1 lỗ hổng web cho phép kẻ tấn công lôi kéo người dùng làm những việc họ không muốn. Nó cho phép kẻ tấn công phá vỡ chính sách chung nguồn gốc, được thiết kế để các trang web khác nhau không can thiệp.

**Những ảnh hưởng của cuộc tấn công CSRF:**

Trong 1 cuộc tấn công CSRF thành công, kẻ tấn công làm cho nạn nhân thực hiện những hành động họ không mong muốn như thay đổi địa chỉ email, mật khẩu và chuyển tiền. Như vậy, kẻ tấn công có thể giành quyền kiểm soát tài khoản của người dùng. Nếu nạn nhân có đặc quyền cao trong ứng dụng, thì kẻ tấn công có thể kiểm soát tất cả chức năng và dữ liệu của ứng dụng luôn.

**Cách CSRF hoạt động?**

Để tấn công CSRF thành công, phải có 3 điều kiện quan trọng sau:

Chức năng leo thang: Ứng dụng phải có chức năng quan trọng để kẻ tấn công muốn lôi kéo như sửa đổi đặc quyền cho người dùng khác hoặc thay đổi mật khẩu

Xử lí phiên dựa trên cookies: khi thực hiện 1 hay nhiều yêu cầu http, ứng dụng dựa vào phiên của cookies để xác thực người dùng đã gửi yêu cầu. Nó không có cơ chế theo dõi phiên hay xác thực yêu cầu của người dùng.

Những tham số của yêu cầu giá trị của nó kẻ tấn công phải dễ xác định. Ví dụ, khi bắt nạn nhân thay đổi mật khẩu, chức năng này không dễ khai thác nếu kẻ tấn công cần biết mật khẩu hiện tại.

Ví dụ, giả sử 1 ứng dụng có chức năng cho phép người dùng thay đổi địa chỉ email. Khi người dùng thực hiện, nó sẽ gửi yêu cầu http 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

Rõ ràng nó đã đáp ứng những điều kiện cần thiết của CSRF:

* Việc thay đổi địa chỉ email rất hứng thú với kẻ tấn công. Kẻ tấn công có thể cập nhật lại mật khẩu và giành quyền kiểm soát tài khoản của nạn nhân.
* Ứng dụng sử dụng cookies để xác thực người dùng. Không có cơ chế theo dõi phiên.
* Kẻ tấn công dễ xác định giá trị của những tham số yêu cầu của chức năng.

Như vậy, kẻ tấn công có thể xây dựng 1 trang web chứa 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 nạn nhân lướt trang web của kẻ tấn công, điều sau đây sẽ xảy ra:

1. Trang của kẻ tấn công sẽ gửi yêu cầu http đến trang web có lỗ hổng.
2. Nếu người dùng đang đăng nhập trang web có lỗ hổng, trình duyệt của họ sẽ tự động thêm cookies trong yêu cầu.
3. Trang web có lỗ hổng sẽ xử lí yêu cầu bình thường 🡪 đổi địa chỉ email thành công.

Mặc dù CSRF được mô tả dựa trên cookies để quản lí phiên, nó cũng sinh ra ở những bối cảnh khác nhau khi ứng dụng tự động thêm thông tin đăng nhập trong yêu cầu như Basic authentication và xác thực dựa trên chứng chỉ.

**Làm sao để tấn công CSRF?**

Việc tạo HTML thủ công để khai thác CSRF có thể rất cồng kềnh vì những yêu cầu rất kì quặc hay chứa nhiều tham số.

Cách dễ nhất để tấn công CSRF là dùng CSRF PoC generator trong Burp pro.

Cơ chế của CSRF rất giống với reflected XSS. Nhìn chung, kẻ tấn công sẽ đặt HTML độc hại trên trang web chúng kiểm soát, sau đó lôi kéo nạn nhân lướt trang đó. Bằng cách gửi người dùng 1 đường dẫn qua email hay tin nhắn mạng xã hội, bình luận, bọn họ chỉ đợi người dùng click vào.

**Lưu ý:** 1 vài khai thác CSRF đơn giản dùng phương thức GET và có thể rất khép kín với 1 url duy nhất trên trang web có lỗ hổng. Trong tình huống đó, kẻ tấn công không cần triển khai trang web bên ngoài và trực tiếp gửi url độc hại. Trong ví dụ trước, nếu chức năng thay đổi email có thể được thực hiện với phương thức GET, thì tấn công sẽ như sau:

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

**XSS vs CSRF:**

Sự khác nhau giữa XSS và CSRF:

XSS cho phép kẻ tấn công thực thi code Javascript trong trình duyệt của nạn nhân.

CSRF cho phép kẻ tấn công lôi kéo nạn nhân làm những hành động họ không muốn.

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

CSRF chỉ áp dụng cho những hành động nhỏ mà người dùng thực hiện được. Rất nhiều ứng dụng web phòng thủ CSRF nhưng vẫn sót 1, 2 hành động. Ngược lại, khai thác XSS thành công có thể khiến người dùng thực hiện bất kỳ hành động nào mà người dùng có thể thực hiện.

CSRF có thể được mô tả là lỗ hổng 1 chiều, khi kẻ tấn công có thể khiến nạn nhân gửi yêu cầu http nhưng họ không thể truy suất phản hồi từ yêu cầu đó. Ngược lại, XSS là hai chiều, khi tập lệnh kẻ tấn công tiêm vào có thể gửi bất cứ yêu cầu nào, đọc phản hồi, và trích xuất dữ liệu đến tên miền bên ngoài mà hắn chọn.

Mã CSRF có thể ngăn chặn cuộc tấn công XSS?

Một vài cuôc tấn công XSS có thể được ngăn chặn qua việc sử dụng hiệu quả mã CSRF. Xem xét 1 lỗ hổng reflected XSS có thể được khai thác như sau:

https://insecure-website.com/status?

message=<script>/\*+Bad+stuff+here...+\*/</script>

Giả sử chức năng này có mã CSRF:

https://insecure-website.com/status?csrftoken=CIwNZNlR4XbisJF39I8yWnWX9wX4Wfoz

&message=<script>/\*+Bad+stuff+here...+\*/</script>

Giả sử máy chủ xác thực mã CSRF, và từ chối những yêu cầu không có mã hợp lệ, thì nó sẽ ngăn chặn luôn khai thác XSS.

Mấu chốt ở đây là “cross site scripting”: thực thi lệnh ở bên ngoài. Khi reflected lại liên quan đến yêu cầu bên ngoài. Bằng cách ngăn chặn kẻ tấn công yêu cầu từ bên ngoài, ứng dụng sẽ ngăn chặn luôn lỗ hổng XSS.

Một số cảnh báo quan trọng:

Nếu lỗ hổng reflected XSS có ở bất cứ đâu trên trang web trong chức năng không được bảo vệ bởi mã CSRF thì XSS vẫn có thể khai thác bình thường.

Nếu lỗ hổng XSS tồn tại, nó có thể được tận dụng để làm cho nạn nhân thực hiện những hành động kể cả nó đã được bảo vệ bởi mã CSRF. Trong tình huống đó, tập lệnh của kẻ tấn công có thể yêu cầu trang liên quan để lấy mã CSRF hợp lệ.

Mã CSRF không chống lại lỗ hổng Stored XSS được vì điểm mấu chốt để khai thác XSS khi có mã CSRF bảo vệ là có được mã hợp lệ.

**CSRF Tokens:**

**Khái niệm:**

Mã CSRF là giá trị duy nhất, bí mật và không thể đoán trước được tạo ra bởi máy chủ và truyền qua máy khách. Nó sẽ được chứa trong yêu cầu http tiếp theo được tạo bởi máy khách. Khi tạo 1 yêu cầu sau đó, ứng dụng sẽ xác thực xem yêu cầu có chứa mã hợp lệ không.

Mã CSRF có thể ngăn chặn cuộc tấn công CSRF, nhờ đó kẻ tấn công không thể tạo 1 yêu cầu http hợp lệ gửi cho nạn nhân được. Vì kẻ tấn công không thể xác định được mã đó, yêu cầu http sẽ không đủ tham số.

Mã CSRF nên chứa significant entropy và khó dự đoán, rất giống với mã phiên.

**Mã CSRF nên được truyền như thế nào?**

Mã CSRF là bí mật nên phải được xử lí an toàn.

1 cách hiệu quả là truyền mã đến máy khách trong mục ẩn của biểu mẫu HTML sẽ được gửi đi bằng phương thức POST. Mã sẽ được chứa trong tham số của yêu cầu khi biểu mẫu được gửi.

<input type="hidden" name="csrf-token" value="CIwNZNlR4XbisJF39I8yWnWX9wX4WFoz" />

Để thêm an toàn, mục chứa mã CSRF nên được đặt sớm nhất trong tài liệu HTML, trước những thẻ input không ẩn và trước những vị trí mà dữ liệu do người dùng kiểm soát được nhúng trong HTML. Nó sẽ giảm thiểu rất nhiều kĩ thuật mà kẻ tấn công có thể tận dụng tài liệu HTML và thu thập nội dung của nó.

Một cách tiếp cận thay thế là đặt token trong chuỗi truy vấn của Url, hơi kém an toàn vì chuỗi truy vấn:

* Được log lại ở nhiều vị trí khác nhau phía máy chủ và máy khách
* Được truyền qua bên thứ ba trong http referer header và có thể hiển thị trong trình duyệt của người dùng.

Một vài ứng dụng truyền mã CSRF trong những headers tùy chỉnh. Nó sẽ bảo mật tốt hơn khi kẻ tấn công có thể dự đoán được hoặc thu thập mã của người dùng vì trình duyệt không cho phép những headers tùy chỉnh được gửi qua tên miền khác. Tuy nhiên, cách tiếp cận này giới hạn ứng dụng tạo những yêu cầu bảo vệ CSRF bằng XHR ( trái ngược với các biểu mẫu HTML) và có thể rất phức tạp ở nhiều tình huống.

Mã CSRF không nên truyền trong cookies.

**Mã CSRF nên được xác thực như thế nào?**

Khi mã CSRF được tạo ra, nó nên được lưu trữ ở phía máy chủ trong phiên của người dùng. Khi yêu cầu tiếp theo được nhận cần xác thực, ứng dụng nên xác thực xem yêu cầu chứa mã đó có đúng với giá trị được lưu trữ trong phiên người dùng không. Việc xác thực bắt buộc phải thực hiện kể cả phương thức http hay loại nội dung của yêu cầu. Nếu yêu cầu không chứa mã CSRF, ứng dụng phải từ chối ngay.

**Ngăn chặn tấn công CSRF:**

Cách phòng thủ mạnh mẽ nhất để chống lại CSRF là chứa mã CSRF trong yêu cầu.

Mã đó nên:

* Khó dự đoán
* Gắn với phiên người dùng
* Được xác thực nghiêm khắc trước khi hành động đó được thực thi.

Một cách phòng thủ hiệu quả khác có thể dùng chung với mã CSRF là Samesite cookies.

**Samesite cookies:**

Một vài trang web phòng thủ CSRF bằng SameSite cookies.

Thuộc tính SameSite có thể dùng để kiểm soát xem cookies được gửi trong những yêu cầu khác miền như thế nào. Bằng việc thiết lập thuộc tính trên cookies, 1 ứng dụng có thể ngăn chặn trình duyệt tự động thêm cookies vào những yêu cầu bất kể nguồn gốc của nó.

Thuộc tính SameSite được thêm vào Set-Cookie response header khi máy chủ phát hành cookies và thuộc tính có thể được nhận 2 giá trị Strict hoặc Lax. Ví dụ:

Set-Cookie: SessionId=sYMnfCUrAlmqVVZn9dqevxyFpKZt30NN; SameSite=Strict;

Set-Cookie: SessionId=sYMnfCUrAlmqVVZn9dqevxyFpKZt30NN; SameSite=Lax;

Nếu thuộc tính SameSite là strict thì trình duyệt sẽ không chứa cookie trong những yêu cầu có nguồn gốc từ 1 trang web khác. Đây là lựa chọn phòng thủ tốt nhất, nhưng nó có thể làm giảm trải nghiệm của người dùng vì nếu người dùng đã đăng nhập theo 1 bên thứ ba đến 1 trang web thì họ sẽ cần đăng nhập lại trước khi tương tác với nó.

Nếu thuộc tính SameSite là lax thì trình duyệt sẽ chứa cookies trong những yêu cầu có nguồn gốc từ trang web khác nếu thỏa 2 điều kiện sau:

* Yêu cầu sử dụng phương thức Get. Những yêu cầu với phương thức khác như Post sẽ không chứa cookies
* Yêu cầu xuất phát từ sự chuyển hướng của người dùng như nhấn vào đường dẫn. Những yêu cầu khác như tập lệnh sẽ không chứa cookies.

Việc sử dụng Cookies ở chế độ Lax chỉ bảo vệ 1 phần các cuộc tấn công CSRF vì những hành động của người dùng là mục tiêu của cuộc tấn công CSRF thường được thực thi bằng phương thức POST.

Hai điều quan trọng là:

* Một vài ứng dụng thực thi chức năng nhạy cảm bằng phương thức GET.
* Rất nhiều ứng dụng và thư viện chấp nhận nhiều phương thức khác nhau. Trong tình huống đó, nếu ứng dụng cố triển khai phương thức POST thì nó vẫn chấp nhận những yêu cầu sử dụng phương thức GET.

Với những lí do trên, không nên chỉ dựa vào SameSite cookies để phòng thủ các cuộc tấn công CSRF. Nên dùng chung với mã CSRF, SameSite cookies có thể cung cấp thêm 1 lớp phòng thủ để giảm thiểu những điểm yếu khác của mã.

**Các lỗ hổng CSRF phổ biến:**

Đa số các lỗ hổng CSRF sinh ra vì những sai xót trong việc xác thực mã CSRF.

Trong ví dụ trước, giả sử 1 ứng dụng web chứa mã CSRF trong yêu cầu thay đổi email của người dùng:

POST /email/change HTTP/1.1

Host: vulnerable-website.com

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

Content-Length: 68

Cookie: session=2yQIDcpia41WrATfjPqvm9tOkDvkMvLm

csrf=WfF1szMUHhiokx9AHFply5L2xAOfjRkE&email=wiener@normal-user.com

Nó phải ngăn chặn được các cuộc tấn công CSRF vì đã vi phạm những điều kiện cần thiết của lỗ hổng này: Ứng dụng không còn phụ thuộc vào cookies để xử lí phiên nữa và yêu cầu chứa tham số giá trị của nó kẻ tấn công không xác định được. Tuy vậy, có rất nhiều cách để lớp phòng thủ này bị phá vỡ 🡪 ứng dụng bị dính lỗi CSRF.

**Việc xác thực mã CSRF phụ thuộc vào phương thức của yêu cầu:**

Một vài ứng dụng xác thực mã khi yêu cầu dùng phương thức POST nhưng bỏ qua xác thực khi phương thức GET được dùng.

Trong tình huống đó, kẻ tấn công có thể điều chỉnh 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 CSRF dựa vào sự hiện diện của mã**

Một vài ứng dụng web xác thực mã khi nó có và bỏ qua nếu mã rỗng.

Trong tình huống đó, kẻ tấn công có thể bỏ toàn bộ tham số chứa mã để bỏ qua xác thực và 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

**Mã CSRF không gắn với phiên làm việc của người dùng**

Một vài ứng dụng không xác thực việc mã CSRF có thuộc về phiên của người dùng đã gửi yêu cầu không. Thay vào đó, ứng dụng duy trì 1 nhóm mã chung để phát hành và chỉ chấp nhận những mã đó.

Trong tình huống đó, kẻ tấn công có thể đăng nhập vào ứng dụng bằng tài khoản của mình, nhận 1 mã hợp lệ, sau đó gửi mã cho nạn nhân để tấn công CSRF.

**Mã CSRF gắn với cookies không phải là phiên:**

Là sự biến thể dựa trên lỗ hổng trước, một vài ứng dụng web gán mã CSRF cho cookie nhưng không phải là cookie theo dõi phiên. Điều này rất dễ xảy ra khi ứng dụng triển khai hai thư viện khác nhau, một cái để xử lí phiên và một cái để bảo vệ CSRF, chúng không tích hợp cùng nhau.

POST /email/change HTTP/1.1

Host: vulnerable-website.com

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

Content-Length: 68

Cookie: session=pSJYSScWKpmC60LpFOAHKixuFuM4uXWF; csrfKey=rZHCnSzEp8dbI6atzagGoSYyqJqTz5dv

csrf=RhV7yQDO0xcq9gLEah2WVbmuFqyOq7tY&email=wiener@normal-user.com

Tình huống này rất khó khai thác nhưng vẫn có lỗ hổng. Nếu trang web chứa bất cứ hành vi nào cho phép kẻ tấn công đặt cookie trong trình duyệt nạn nhân mới có thể tấn công. Kẻ tấn công có thể đăng nhập ứng dụng, nhận 1 mã hợp lệ và cookies tương ứng, tận dụng hành vi đặt lại cookie để đặt lại cookie của họ trong trình duyệt nạn nhân và gửi mã để tấn công CSRF.

**Mã CSRF được lặp lại ở cookie:**

Một biến thể nữa của lỗ hổng trước, một vài ứng dụng web không duy trì những mã phía máy chủ đã phát hành, thay vào đó sẽ lặp lại mỗi mã trong cookie và tham số. Khi yêu cầu tiếp theo được gửi đi, ứng dụng chỉ đơn giản xác thực xem token được gửi đi trong tham số có giống với giá trị của cookie.

POST /email/change HTTP/1.1

Host: vulnerable-website.com

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

Content-Length: 68

Cookie: session=1DQGdzYbOJQzLP7460tfyiv3do7MjyPw; csrf=R8ov2YBfTYmzFyjit8o2hKBuoIjXXVpa

csrf=R8ov2YBfTYmzFyjit8o2hKBuoIjXXVpa&email=wiener@normal-user.com

Trong tình huống đó, kẻ tấn công có thể tấn công CSRF nếu trang web có chức năng chỉnh sửa cookie. Ở đây, kẻ tấn công không cần nhận mã hợp lệ của họ. Họ chỉ đơn giản tạo ra mã (có thể ở định dạng bắt buộc), tận dụng hành vi chỉnh sửa cookie để đặt cookie trong trình duyệt nạn nhân và gửi mã để tấn công CSRF.

**Phòng thủ dựa vào Referer để chống lại CSRF:**

Ngoài mã CSRF, một vài ứng dụng dùng Referer header để phòng thủ các cuộc tấn công CSRF, bằng việc xác thực xem yêu cầu có nguồn gốc từ tên miền của ứng dụng không.

Trong tình huống đó, kẻ tấn công có thể khai thác CSRF theo cách khiến cho trình duyệt nạn nhân bỏ qua Referer header. Cách dễ nhất là dùng thẻ META trong trang HTML lưu trữ cho cuộc tấn công CSRF.

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

**Xác thực dựa trên Referer có thể bị phá vỡ:**

Một vài ứng dụng xác thực Referer header 1 cách ngây thơ có thể bị bỏ qua. Ví dụ, nếu ứng dụng xác thực tên miền trong Referer bắt đầu với giá trị mong đợi thì kẻ tấn công có thể đặt nó như tên miền con trong tên miền của chúng

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

Tương tự như vậy, nếu ứng dụng đơn giản xác thực Referer chứa tên miền của nó thì kẻ tấn công có thể đặt giá trị yêu cầu ở bất cứ đâu trong Url:

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

**Lưu ý:** Mặc dù bạn có thể xác thực hành vi này bằng Burp, bạn sẽ thường thấy cách tiếp cận này không còn hoạt động nữa khi bạn kiểm tra nó trên trình duyệt. Trong 1 nỗ lực nhằm giảm rủi ro dữ liệu nhạy cảm bị rò rỉ theo cách này, rất nhiều trình duyệt đã bỏ qua chuỗi truy vấn từ header Referer.

Bạn có thể ghi đè hành vi này bằng việc đảm bảo rằng phản hổi có chứa khai thác của bạn có Referrer-Policy: unsafe-url header (chú ý là Referrer phải viết đúng chính tả trong trường hợp này). Nó đảm bảo full Url sẽ được gửi đi, bao gồm cả chuỗi truy vấn.