## **What is CSRF?**

Cross-site request forgery (also known as CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.

## **What is the impact of a CSRF attack?**

In a successful CSRF attack, the attacker causes the victim user to carry out an action unintentionally. For example, this might be to change the email address on their account, to change their password, or to make a funds transfer. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality.

A Cross-Site Request Forgery (CSRF) attack tricks a user into performing an action they didn’t intend to on a website they’re authenticated with. By exploiting the trust a website has in the user’s browser, CSRF lets an attacker make requests on the user’s behalf without their knowledge.

Here’s how CSRF works and the conditions that make it possible:

1. \*\*A Relevant Action\*\*

For a CSRF attack to be meaningful, the attacker needs a valuable action they want to trigger. This action might change something on the server, like updating account details, transferring money, or adjusting settings. The attacker wants to force the authenticated user to execute this action without realizing it.

2. \*\*Cookie-Based Session Handling\*\*

CSRF relies on session cookies, which websites use to remember users between requests. When you log in, the website creates a session, and your browser stores a session cookie. Every subsequent request to the site sends this cookie automatically, letting the server know you’re authenticated. CSRF attacks exploit this mechanism by getting the browser to send the session cookie on a malicious request to the site. Importantly, there’s no secondary session or authentication check (such as a unique token) involved, so the site trusts the request.

3. \*\*No Unpredictable Request Parameters\*\*

For CSRF to succeed, the attacker needs to craft a request with predictable parameters. For instance, if the attacker wants to make the user change their password, they need to know the structure of the password-change request and, potentially, the current password. If the application requires an unpredictable parameter, like a randomly generated token in the request, this would prevent the attacker from guessing or forging a valid request.

In summary, CSRF attacks depend on:

- A valuable action the attacker wants to exploit.

- Session cookies alone for user authentication.

- Predictable parameters that don’t require specific user knowledge.

## Mitigations usually include using tokens (like CSRF tokens) in forms, requiring re-authentication for sensitive actions, and implementing anti-CSRF libraries. These prevent attackers from sending requests that look like they’re coming from the legitimate user’s session. **How does CSRF work? - Continued**

For example, suppose an application contains a function that lets the user change the email address on their account. When a user performs this action, they make an HTTP request like the following:

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

This meets the conditions required for CSRF:

* The action of changing the email address on a user's account is of interest to an attacker. Following this action, the attacker will typically be able to trigger a password reset and take full control of the user's account.
* The application uses a session cookie to identify which user issued the request. There are no other tokens or mechanisms in place to track user sessions.
* The attacker can easily determine the values of the request parameters that are needed to perform the action

With these conditions in place, the attacker can construct a web page containing the following HTML:

<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>  
  
If a victim user visits the attacker's web page, the following will happen:

* The attacker's page will trigger an HTTP request to the vulnerable website.
* If the user is logged in to the vulnerable website, their browser will automatically include their session cookie in the request (assuming SameSite cookies are not being used).
* The vulnerable website will process the request in the normal way, treat it as having been made by the victim user, and change their email address.

#### **Note**

Although CSRF is normally described in relation to cookie-based session handling, it also arises in other contexts where the application automatically adds some user credentials to requests, such as HTTP Basic authentication and certificate-based authentication.