<https://brightsec.com/blog/cross-site-request-forgery-csrf/>

<https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html>

What is it

* An attack that sends a request to the server on the user’s behalf without them intending to
* If a website is only using POST requests, you can’t frame malicious requests using an <a> tag, but it can be delivered in a <form> tag.
* Does not return data to the attacker
* Examples:
  + Change email / password
  + Transfer funds
  + Modify applications if the user is privileged

How does it work

The attacker forges a request for the website

Then the attacker embeds said request into links and sends the link out

If the user is logged into the site, clicking the link will send the request to the server, where the action is performed

Cookie-based session handling

* The action requires issuing at least one HTTP request
* The application relies on session cookies to identify the user who has made requests
* No other mechanism for tracking sessions or validating user requests can be in place

No unpredictable request parameters

* The HTTP requests cannot contain parameters that the attacker cannot determine or guess the values of, such as causing a user to change their password if the existing password is required

Prevention

* XSS can default ALL CSRF mitigation techniques according to owasp
* Verify the origin with standard headers
* Do not use GET requests for state changing operations
* Many frameworks have

CSRF Tokens

* A unique, secret, and unpredictable value generated by the server-side application and shared with the client
* The client must include the token in the request for any sensitive actions
* This prevents attackers from forming fully valid HTTP requests
* Should be added to any state-changing or action-causing requests and validated on the backend
* Insert into a custom HTTP request header via JavaScript

Custom Request Headers for APIs

* No tokens required
* Client appends a custom header to requests that require CSRF protection
* The header can be anything if it does not conflict with any existing headers
* The API checks if the header exists. If it does not, the backend rejects the request

Simple Requests

A simple request is permitted to be sent to any origin by the browser, introducing CSRF risks. To be deemed simple, a request must have one of the following content types: application/x-www-form-urlencoded, multipart/form-data or text/plain. An example of a simple request is using a <form> tag to submit data. Any simple requests should be protected with CSRF prevention methods. Another solution would be to disallow these simple content types.

CSRF Token Vulnerabilities

**Validation depends on presence of token**

Some applications skip the verification process if the token doesn’t exist. If the attacker finds the code containing the token information and removes it, the token validation is skipped.

**CSRF token is not associated with user session**

Some applications keep a pool of tokens, and if any token from the pool is used, it is accepted. This allows attackers to impersonate any user if they have at least one token from the pool.

**Token validation changes with HTTP method**

In some applications, using the GET method instead of the POST method will cause CSRF validation not to work properly. If the attacker manages to switch the method, the verification process will be bypassed.

**CSRF token is copied to the cookie**

Some applications do not keep a record of tokens already in use, rather they copy the request parameters associated with each token into the user’s cookie. This means the attacker can create a cookie containing a token that uses the application’s expected format and place it in the user’s browser, then execute a CSRF attack. This request will be validated because it will match the malicious cookie.

SameSite Cookie Attribute

* Tells the browser when it is okay to send cookies with cross-site requests
* Shouldn’t be set specifically for a domain, as all subdomains will share the cookie.

User Interaction Based Defense

* Can harm UI, but is appropriate in many circumstances, especially when dealing with sensitive information such as financial details.
* Examples include CAPTCHA and one-time passwords sent to a user’s email or phone number

Client-Side CSRF Attacks

* Attacker tricks the client-side JavaScript code to send an HTTP request to a vulnerable target site by manipulating the program’s input parameters
* Can bypass token-based mitigations and SameSite cookies, as the JavaScript program will include things such as CSRF tokens or custom headers in the requests
* Web browsers will include cookies in same-site request contexts initiated by JavaScript programs, circumventing SameSite cookie policies

Code Examples

Real World Example