Cross Site Request Forgery protection[**¶**](https://docs.djangoproject.com/en/2.1/ref/csrf/#module-django.middleware.csrf)

The CSRF middleware and template tag provides easy-to-use protection against [Cross Site Request Forgeries](https://www.squarefree.com/securitytips/web-developers.html#CSRF). This type of attack occurs when a malicious website contains a link, a form button or some JavaScript that is intended to perform some action on your website, using the credentials of a logged-in user who visits the malicious site in their browser. A related type of attack, ‘login CSRF’, where an attacking site tricks a user’s browser into logging into a site with someone else’s credentials, is also covered.

## Cross-site request forgery (CSRF)

A Cross-site request forgery hole is when a malicious site can cause a visitor's browser to make a request to your server that causes a change on the server. The server thinks that because the request comes with the user's cookies, the user wanted to submit that form.

Depending on which forms on your site are vulnerable, an attacker might be able to do the following to your victims:

* Log the victim out of your site. (On some sites, "Log out" is a link rather than a button!)
* Change the victim's site preferences on your site. ([Example: Google](http://www.squarefree.com/2003/10/23/another-google-security-hole))
* Post a comment on your site using the victim's login.
* Transfer funds to another user's account.

Attacks can also be based on the victim's IP address rather than cookies:

* Post an anonymous comment that is shown as coming from the victim's IP address.
* Modify settings on a device such as a wireless router or cable modem.
* Modify an intranet wiki page.
* Perform a distributed password-guessing attack without a botnet. (This assumes they have a way to tell whether the login succeeded, perhaps by submitting a second form that isn't protected against CSRF.)

CSRF attacks usually involve JavaScript to submit the cross-site form automatically. It is possible for a malicious site to make a user submit a form to another site even without JavaScript, however: form fields can be hidden and buttons can be disguised as links or scrollbars.

### Preventing CSRF

Make sure form submissions that cause server-side changes use your own forms. There are two ways you can do this:

1. Check the referrer header. If it is not present, or if it does not show the correct URL as the referrer, reject the submission. This has the advantage of being simple and sane, but the disadvantage that users who have told their browsers to omit the referrer header (out of concern for privacy) or lie about the referrer (in order to gain access to porn sites that use the referrer for the wrong purpose) will have trouble. This strategy doesn't work if the form uses GET and the page can contain user-generated content with links.
2. Include a hidden field in the form and check its value when the form is submitted. A simple scheme is to use an MD5 hash of the login cookie, some information about the form, and a secret on the server. Another possibility is to use a randomly generated one-time key for every form you serve, assuming you have a sufficiently unpredictable [source of random numbers](http://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator). This has the disadvantage of making it unsafe for users to save the HTML for forms and upload them to, say, bug databases.