Python/Django Cross Site Request Forgery protection.

The CSRF middleware and template tag provides easy-to-use protection against Cross Site Request Forgeries.

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.

The first defense against CSRF attacks is to ensure that GET requests (and other ‘safe’ methods, as defined by RFC) are side effect free.

Requests via ‘unsafe’ methods, such as POST, PUT, and DELETE, can then be protected by following the steps below.

How to use it:

To take advantage of CSRF protection in your views, follow these steps:

1. The CSRF middleware is activated by default in the MIDDLEWARE setting. If you override that setting, remember that 'django.middleware.csrf.CsrfViewMiddleware' should come before any view middleware that assume that CSRF attacks have been dealt with. If you disabled it, which is not recommended, you can use csrf\_protect() on particular views you want to protect.

2. In any template that uses a POST form, use the csrf\_token tag inside the <form> element if the form is for an internal URL, e.g.:

<form method="post">{% csrf\_token %}

This should not be done for POST forms that target external URLs, since that would cause the CSRF token to be leaked, leading to a vulnerability.

3. In the corresponding view functions, ensure that RequestContext is used to render the response so that {%csrf\_token %} will work properly. If you’re using the render() function, generic views, or contrib apps, you are covered already since these all use RequestContext.

Compiled and presented by Vakindu Philliam.