CSRF protection and ezformtoken

How cross-site request forgery is prevented in Exponential and what PHP code must do - the ezformtoken listeners on request/input and response/output, which requests are checked, the form field and the X-CSRF-Token...

Source file: `doc/eng-us/help/php-developers/csrf-and-ezformtoken.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/php-developers/csrf-and-ezformtoken.md

This page is for PHP developers whose code receives POST requests. Cross-site request forgery makes a signed-in user's browser send a request the user did not intend, from a hostile page. Exponential blocks it with the `ezformtoken` extension, which is part of the kernel distribution (`extension/ezformtoken`). Most code is protected without doing anything; this page explains when that holds and what to do when it does not.

Contents

1. How it works
2. Which requests are checked
3. What your code must do
4. POSTs from JavaScript
5. Scripts and APIs
6. The refusal
7. Settings
8. Mistakes to avoid

How it works

`extension/ezformtoken/settings/site.ini.append.php` registers three listeners of the kernel's event hub:

INI

```ini
[Event]
Listeners[]=request/input@ezxFormToken::input
Listeners[]=response/output@ezxFormToken::output
Listeners[]=session/regenerate@ezxFormToken::reset
```

Listener	Does
`ezxFormToken::output()`	adds a hidden field `ezxform_token` with the user's token to every `method="post"` form of the page
`ezxFormToken::input()`	checks every POST before any module runs; a missing or wrong token ends the request
`ezxFormToken::reset()`	gives the session a new token when the session id changes (sign-in)

`ezxFormToken::getToken()` returns it: a SHA-1 hash of the secret, the intention and the session id.

Which requests are checked

`ezxFormToken::input()` checks a request when all of these hold:

- it is a POST (or carries POST data);
- the session has started and the user is **signed in** (anonymous POSTs are not checked: an anonymous visitor has no session to abuse);
- it is not a Content Security Policy report.

The token is read from the POST field (`ezxform_token`, or the field set with `setFormField()`), or from the header `X-CSRF-Token`, which lets JavaScript send JSON or XML bodies.

What your code must do

1. **Change data only on POST.** A GET that changes data cannot be protected; a link or an image can trigger it.
2. **Render forms through templates** with `method="post"`; the token is added for you.
3. **Use the module's actions** (`$Module->isCurrentAction()`) or `eZHTTPTool` to read the POST; by the time your code runs, the token was checked.
4. **Do not switch the check off** for your module. `ezxFormToken::setIsEnabled( false )` exists for special cases in the kernel; it is not a fix for a failing form.

POSTs from JavaScript

JavaScript that posts with `fetch()` sends the token, read from the page:

JavaScript

```
// the token is in every form the page has; or render it into a data attribute of your own
var token = document.querySelector( 'input[name="ezxform_token"]' ).value;
fetch( url, { method: 'POST', credentials: 'same-origin',
              headers: { 'X-CSRF-Token': token, 'Content-Type': 'application/json' },
              body: JSON.stringify( data ) } );
```

When the page has no form, render the token into a `data-*` attribute in your template; in PHP, `ezxFormToken::getToken()` returns it. Do not print it into an inline script (Escaping output from PHP).

Scripts and APIs

Programs are not browsers and do not use form tokens:

- The REST API authenticates with OAuth2 tokens or API keys (REST and remote services from PHP).
- expservices accepts a personal API token in `Authorization: Bearer` or `X-Exp-Token`; a request that carries it needs POST for writes but no form token, because a hostile page cannot set that header. A request without the token header falls back to the cookie and the form token (Backend services over ezjscore).
- Command line scripts do not go through the web kernel at all.

The refusal

A refused POST ends with `ezpFormTokenException`: the kernel answers with a 403 page (or a JSON body for XHR and JSON requests) and writes one line to `var/log/warning.log`. Visitors see a "form expired" page that says nothing was saved and asks them to reload the form and send it again (Form expired page).

Settings

`site.ini [HTMLForms]`:

Key	Meaning
`Secret`	the secret mixed into tokens; set a value of your own per installation (the shipped value says to change it)
`RefusalLogCollapseSeconds`	the same refusal again within this many seconds is counted instead of logged again (default 60; 0 logs every one)

Mistakes to avoid

Mistake	Effect	Fix
a form built in PHP and printed raw, outside the page output filter	no token, every POST refused	render it through a template
a GET link that deletes	forgeable	a POST form with a button
`fetch()` without the token	403 for signed-in users	send `X-CSRF-Token`
catching the refusal to "make the form work"	protection gone	find why the token is missing
the token in a URL	leaks into logs and referrers	POST body or header only

Related pages

- Forms, CSRF protection and HTTP input
- ezformtoken and CSRF reference
- Security in PHP code
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

Become an Exponential Partner

Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.

Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.