**<https://www.formget.com/codeigniter-xss-clean/>**

**The CSRF token is added to the form as a hidden input only when the form\_open() function is used.**

**A cookie with the CSRF token's value is created by the Security class, and regenerated if necessary for each request.**

**If $\_POST data exists, the cookie is automatically validated by the Input class. If the posted token does not match the cookie's value, CI will show an error and fail to process the $\_POST data.**

**So basically, it's all automatic - all you have to do is enable it in your $config['csrf\_protection'] and use the form\_open() function for your form.**

**Security Class**

The Security Class contains methods that help you create a secure application, processing input data for security.

* [XSS Filtering](https://www.codeigniter.com/userguide3/libraries/security.html#xss-filtering)
* [Cross-site request forgery (CSRF)](https://www.codeigniter.com/userguide3/libraries/security.html#cross-site-request-forgery-csrf)
* [Class Reference](https://www.codeigniter.com/userguide3/libraries/security.html#class-reference)

[**XSS Filtering**](https://www.codeigniter.com/userguide3/libraries/security.html#id1)

CodeIgniter comes with a Cross Site Scripting prevention filter, which looks for commonly used techniques to trigger JavaScript or other types of code that attempt to hijack cookies or do other malicious things. If anything disallowed is encountered it is rendered safe by converting the data to character entities.

To filter data through the XSS filter use the xss\_clean() method:

$data **=** $this**->**security**->**xss\_clean($data);

An optional second parameter, *is\_image*, allows this function to be used to test images for potential XSS attacks, useful for file upload security. When this second parameter is set to TRUE, instead of returning an altered string, the function returns TRUE if the image is safe, and FALSE if it contained potentially malicious information that a browser may attempt to execute.

**if** ($this**->**security**->**xss\_clean($file, **TRUE**) **===** **FALSE**)

{

*// file failed the XSS test*

}

**Important**

If you want to filter HTML attribute values, use [html\_escape()](https://www.codeigniter.com/userguide3/general/common_functions.html" \l "html_escape" \o "html_escape) instead!

[**Cross-site request forgery (CSRF)**](https://www.codeigniter.com/userguide3/libraries/security.html#id2)

You can enable CSRF protection by altering your **application/config/config.php** file in the following way:

$config['csrf\_protection'] **=** **TRUE**;

If you use the [*form helper*](https://www.codeigniter.com/userguide3/helpers/form_helper.html), then form\_open() will automatically insert a hidden csrf field in your forms. If not, then you can use get\_csrf\_token\_name() and get\_csrf\_hash()

$csrf **=** **array**(

'name' **=>** $this**->**security**->**get\_csrf\_token\_name(),

'hash' **=>** $this**->**security**->**get\_csrf\_hash()

);

**...**

**<**input type**=**"hidden" name**=**"<?=$csrf['name'];?>" value**=**"<?=$csrf['hash'];?>" **/>**

Tokens may be either regenerated on every submission (default) or kept the same throughout the life of the CSRF cookie. The default regeneration of tokens provides stricter security, but may result in usability concerns as other tokens become invalid (back/forward navigation, multiple tabs/windows, asynchronous actions, etc). You may alter this behavior by editing the following config parameter

$config['csrf\_regenerate'] **=** **TRUE**;

Select URIs can be whitelisted from csrf protection (for example API endpoints expecting externally POSTed content). You can add these URIs by editing the ‘csrf\_exclude\_uris’ config parameter:

$config['csrf\_exclude\_uris'] **=** **array**('api/person/add');

Regular expressions are also supported (case-insensitive):

$config['csrf\_exclude\_uris'] **=** **array**(

'api/record/[0-9]+',

'api/title/[a-z]+'

);

[**Class Reference**](https://www.codeigniter.com/userguide3/libraries/security.html#id3)

***class*CI\_Security**

**xss\_clean(*$str*[, *$is\_image = FALSE*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$str** (*mixed*) – Input string or an array of strings |
| **Returns:** | XSS-clean data |
| **Return type:** | Mixed |

Tries to remove XSS exploits from the input data and returns the cleaned string. If the optional second parameter is set to true, it will return boolean TRUE if the image is safe to use and FALSE if malicious data was detected in it.

**Important**

This method is not suitable for filtering HTML attribute vales! Use **[html\_escape()](https://www.codeigniter.com/userguide3/general/common_functions.html" \l "html_escape" \o "html_escape)** for that instead.

**sanitize\_filename(*$str*[, *$relative\_path = FALSE*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$str** (*string*) – File name/path * **$relative\_path** (*bool*) – Whether to preserve any directories in the file path |
| **Returns:** | Sanitized file name/path |
| **Return type:** | String |

Tries to sanitize filenames in order to prevent directory traversal attempts and other security threats, which is particularly useful for files that were supplied via user input.

$filename **=** $this**->**security**->**sanitize\_filename($this**->**input**->**post('filename'));

If it is acceptable for the user input to include relative paths, e.g. *file/in/some/approved/folder.txt*, you can set the second optional parameter, **$relative\_path** to TRUE.

$filename **=** $this**->**security**->**sanitize\_filename($this**->**input**->**post('filename'), **TRUE**);

**get\_csrf\_token\_name()**

|  |  |
| --- | --- |
| **Returns:** | CSRF token name |
| **Return type:** | string |

Returns the CSRF token name (the **$config['csrf\_token\_name']** value).

**get\_csrf\_hash()**

|  |  |
| --- | --- |
| **Returns:** | CSRF hash |
| **Return type:** | string |

Returns the CSRF hash value. Useful in combination with **get\_csrf\_token\_name()** for manually building forms or sending valid AJAX POST requests.

**entity\_decode(*$str*[, *$charset = NULL*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$str** (*string*) – Input string * **$charset** (*string*) – Character set of the input string |
| **Returns:** | Entity-decoded string |
| **Return type:** | String |

This method acts a lot like PHP’s own native **html\_entity\_decode()** function in ENT\_COMPAT mode, only it tries to detect HTML entities that don’t end in a semicolon because some browsers allow that.

If the **$charset** parameter is left empty, then your configured **$config['charset']** value will be used.

**get\_random\_bytes(*$length*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$length** (*int*) – Output length |
| **Returns:** | A binary stream of random bytes or FALSE on failure |
| **Return type:** | String |

A convenience method for getting proper random bytes via **mcrypt\_create\_iv()**, **/dev/urandom** or **openssl\_random\_pseudo\_bytes()** (in that order), if one of them is available.

Used for generating CSRF and XSS tokens.

**Note**

The output is NOT guaranteed to be cryptographically secure, just the best attempt at that.

[Next](https://www.codeigniter.com/userguide3/libraries/sessions.html)[Previous](https://www.codeigniter.com/userguide3/libraries/parser.html)