Preventing CSRF

CSRF is an abbreviation for cross-site request forgery, where a malicious user tricks the user’s browser  
into silently performing an HTTP request to the website when the user is logged in.

An example of such an attack is inserting an invisible image tag with src pointing to  
http: //example. com/site/logout. Even if the image tag is inserted in another website, you will be  
immediately logged out from example. com. The consequences of CSRF can be very serious: destroying  
website data, preventing all website users from logging in, exposing private data, and so on.

Some facts about CSRF:

* As CSRF should be performed by the victim user’s browser, the attacker cannot normally change  
  the HTTP headers sent. However, there are both browser and Flash plugin vulnerabilities that exist  
  which allow users to spoof headers, so we should not rely on these.
* The attacker should pass the same parameters and values as the user would normally.

Considering these, a good method of dealing with CSRF is by passing and checking a unique token during  
form submissions and, additionally, using GET according to the HTTP specification.

Yii includes built-in token generation and token checking. Additionally, it can automate inserting a token  
into the HTML forms.

In order to avoid CSRF, you should always:

* Follow, HTTP specification, that is, get should not change its application state
* Keep Yii CSRF protection enabled

In this recipe, we will see how to make sure our application is CSRF-resistant.

Getting ready

Create a new application by using the Composer package manager, as described in the official guide at  
[http://www.yiiframework.com/doc-2.0/gurde-start-installation.html](http://www.yiiframework.com/doc-2.0/guide-start-installation.html)**.**

How to do it...

1. In order to turn ON the anti-CSRF protection, we should add config/main. php as follows:  
'components' => [  
request => [

'enableCsrfValidation => true,

].

],

1. The option enableCsrfvalidation defaults to true. When CSRF validation is enabled, forms  
   submitted to a Yii web application must originate from the same application. If not, a 400 http  
   exception will be raised.

Note that this feature requires that the user client accepts cookies.

1. After configuring the application, you should use ActiveForm : : beginForm and CHtml: : endForm  
   instead of HTML form tags in view with ActiveForm:

<?php $form = ActiveForm::begin(['id' => 'login-form']); ?>

<input type='text' name='name'

<?php ActiveForm::end(); ?>

1. OR manually:

<form action='#' method='POST'>

<input type="hidden" name="<?= Yii::$app->request->csrfParam ?>" value="<?

=Yii::$app->request->getCsrfToken()?>" />

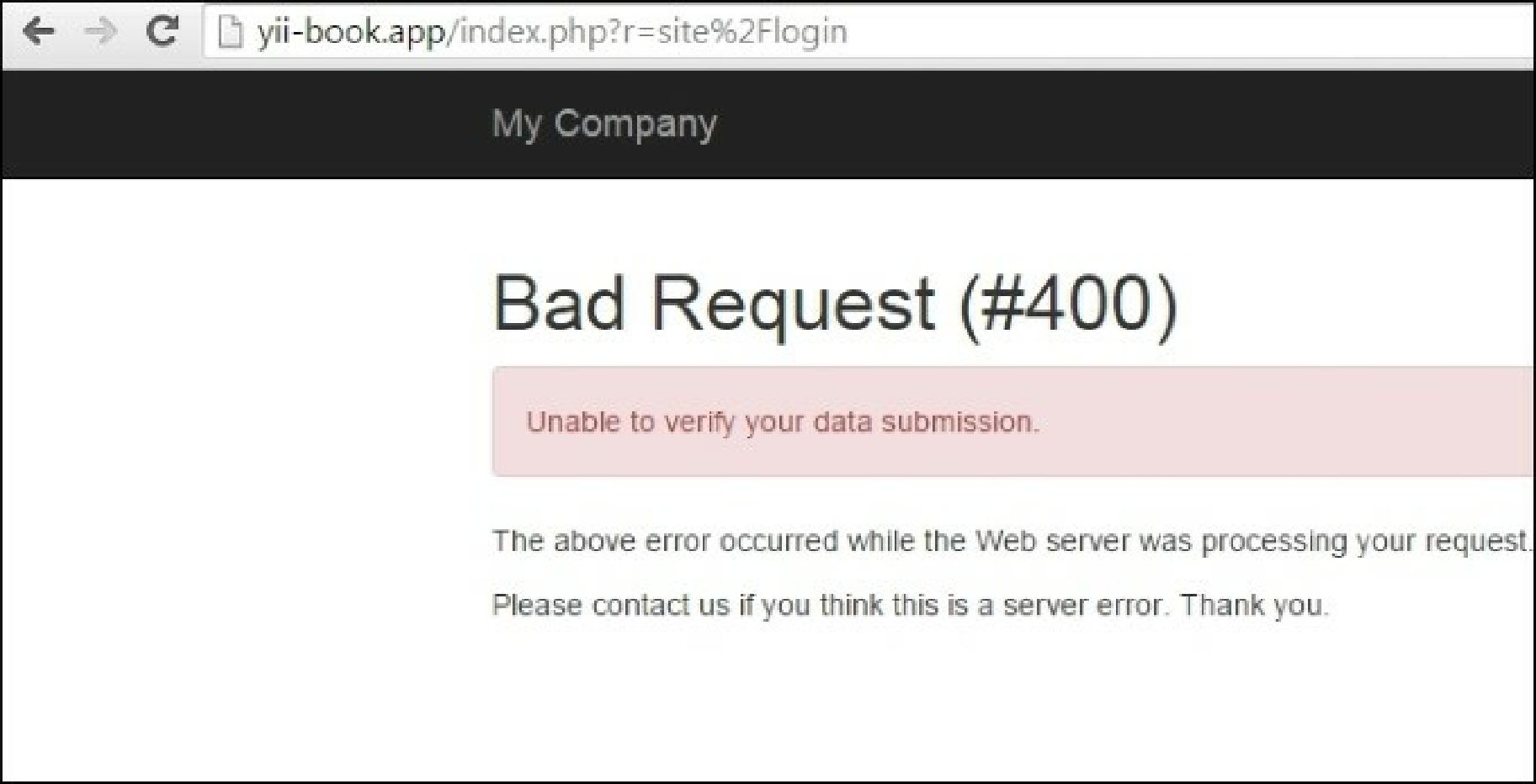
</form>

1. In the first case, Yii automatically adds a hidden token field, as follows:

<form action="/csrf/create" method="post">

<div style="display:none"><input type="hidden" value="e4d1021e79ac  
269e8d6289043a7a8bc154d7115a" name="YII\_CSRF\_TOKEN" />

1. If you save this form as HTML and try submitting it, you will get a message like the one shown in  
   the following screenshot instead of regular data processing:



How it works...

Internally, during form rendering, we have code like this:

if ($request->enableCsrfValidation && !strcasecmp($method, 'post')) {

$hiddenInputs[] = static::hiddenInput($request->csrfParam, $request->getCsrfToken());

}

if (!empty($hiddenInputs)) {

$form .= "\n" . implode("\n", $hiddenInputs);

}

In the previous code, getcsrfToken () generates a unique token value and writes it to a cookie. Then, on  
subsequent requests, both the cookie and post values are compared. If they don’t match, an error  
message is shown instead of usual data processing.

If you need to perform a post request but don’t want to build a form using CHtml, then you can pass a  
parameter with a name from Yii: :app()->request->csrfParam and a value from Yii: :$app->request-  
>getCsrfToken().

There’s more.

Lets have a look at some more features.

Disabling CSRF-tokens for all actions

1. If you have a problem with enableCsrfvalidation you can switch it off.
2. To disable CSRF, add this code to your controller:

public function beforeAction($action) {

$this->enableCsrfValidation = false;  
return parent::beforeAction($action);

}

Disabling CSRF-tokens for a specific action

public function beforeAction($action) {

$this->enableCsrfValidation = ($action->id !== "actionId");  
return parent::beforeAction($action);

}

CSRF validation for Ajax-calls

When the enableCsrfValidation option is enabled in the main layout, add csrfMetaTags:

<head>

<?= Html::csrfMetaTags() ?>

</head>

Now you will be able to simply add it to ajax-call

var csrfToken = $('meta[name="csrf-token"]').attr("content");

$.ajax({

url: 'request'

type: ' post',  
dataType: ' j son',

data: {paraml: paraml, \_csrf : csrfToken},

});

Additionally [rename]

If your application requires a very high security level, such as a bank account management system, extra  
measures can be taken.

First, you can turn off the remember me feature using config/main .php, as follows:

'components' => [

'user' => [

'enableAutoLogin' => false,

].

],

Note that this will not work if the enabledSession option is true.

Then, you can lower the session timeout, as follows:

'components' => [

'session' => [

'timeout' => 200,

],..

],

This sets the number of seconds after which data will be seen as **garbage** and cleaned up.

Of course, these measures will make the user experience worse, but they will add an additional level of  
security.

Using GET and POST properly

HTTP insists on not using get operations that change data or state. Sticking to this rule is good practice. It  
will not prevent all types of CSRF, but it will at least implement some injections, such as <img src=,  
pointless>.

See also

In order to learn more about SQL injections and working with databases through Yii, refer to the following  
URLs:

* [http://en.wikipedia.org/wiki/Cross-site request forgery](http://en.wikipedia.org/wiki/Cross-site_request_forgery)
* [http://www.viiframework.com/doc-2.0/guide-securitv-best-practices.html#avoiding-csrf](http://www.yiiframework.com/doc-2.0/guide-security-best-practices.html%23avoiding-csrf)
* [http://www.yiiframework.com/doc-2.0/yiL-web-rpqupst.html#$enableCsrfValidation-detail](http://www.yiiframework.com/doc-2.0/yii-web-request.html%23%24enableCsrfValidation-detail)
* The *Preventing XSS* recipe.