Предотвращения csrf атак

CSRF-это аббревиатура для подделки межсайтовых запросов, где злонамеренный пользователь обманывает браузер пользователя, чтобы он молча выполнял HTTP-запрос на веб-сайт, когда пользователь вошел в систему.

Примером такой атаки является вставка невидимого тега изображения с src, указывающим на пример http://example.com/site/logout. Даже если тег изображения вставлен в другой веб-сайт, вы сразу же выйдете из example.com. Последствия CSRF могут быть очень серьезными: уничтожение данных сайта, предотвращение всех пользователей сайта от входа в систему, подвергая частные данные, и так далее.

Некоторые факты о CSRF:

• Поскольку CSRF должен выполняться браузером пользователя-жертвы, злоумышленник обычно не может изменить отправленные заголовки HTTP. Тем не менее, существуют уязвимости как браузера, так и плагина Flash, которые позволяют пользователям подделывать заголовки, поэтому мы не должны полагаться на них.

• Злоумышленник должен передать те же параметры и значения, что и обычно пользователь.

Учитывая это, хорошим методом работы с CSRF является передача и проверка уникального маркера во время отправки формы и, кроме того, использование GET в соответствии со спецификацией HTTP.

Yii включает в себя встроенную генерацию токенов и проверку токенов. Кроме того, он может автоматизировать вставку маркера в HTML-формы.

Во избежание CSRF, вы должны всегда:

• Следуйте cпецификаци HTTP, то есть GET не должен изменять состояние приложения

• Держать в Yii защиту от csrf включеной

В этом рецепте мы увидим, как убедиться, что наше приложение устойчиво к CSRF.

Подготовка

Создайте новое приложение с помощью диспетчера пакетов Composer, как описано в официальном руководстве по адресу  
<http://www.yiiframework.com/doc-2.0/guide-start-installation.html>.   
По русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/start-installation>**.**

Как это сделать...

1. Для того, чтобы включить анти-CSRF защиту, мы должны добавить в config/main.php следующее:  
'components' => [  
request => [

'enableCsrfValidation => true,

].

],

1. . Параметр enableCsrfvalidation по умолчанию равен true. Когда проверка CSRF включена, формы, отправленные в веб-приложение Yii, должны исходить из одного и того же приложения. В противном случае возникнет исключение 400 HTTP.

Обратите внимание, что эта функция требует, чтобы пользователь клиент принимал куки

1. После настройки приложения следует использовать ActiveForm::beginForm и ActiveForm::endForm вместо тегов HTML-форм в представлении ActiveForm:

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

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

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

1. Или вручную:

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

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

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

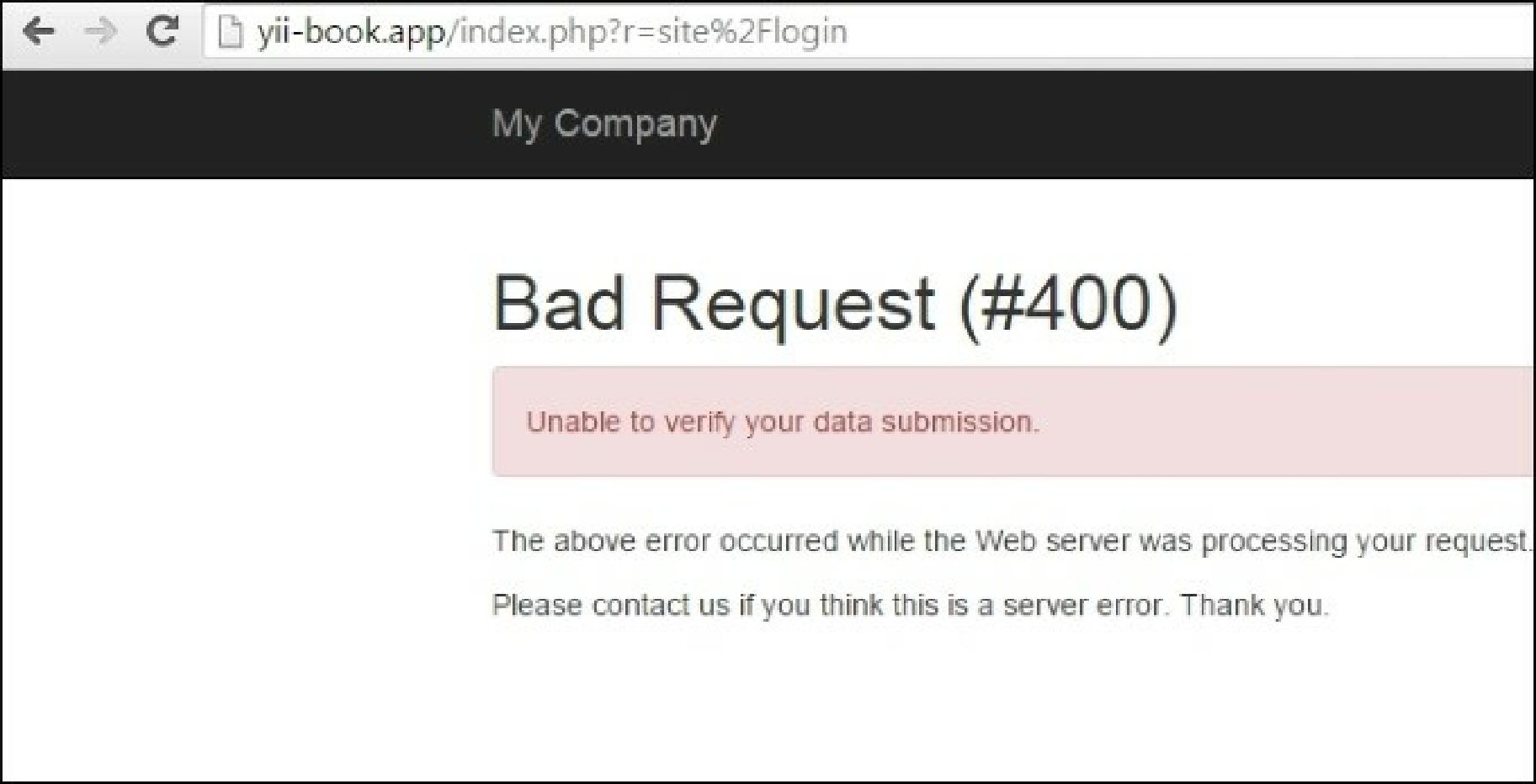
</form>

1. В первом случае Yii автоматически добавляет скрытое поле маркера следующим образом:

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

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

1. Если вы сохраните эту форму как HTML и попытаетесь отправить ее, Вы получите сообщение, подобное показанному на следующем снимке экрана, вместо обычной обработки данных:



Как это работает...

Внутренне, во время рендеринга формы, у нас есть такой код:

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

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

}

if (!empty($hiddenInputs)) {

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

}

В предыдущем коде getcsrfToken() создает уникальное значение токена и записывает его в файл cookie. Затем при последующих запросах сравниваются значения cookie и POST. Если они не совпадают, вместо обычной обработки данных отображается сообщение об ошибке.

Если Вам необходимо выполнить Post запрос, но не хотите создать форму, используя метод CHtml, то вы можете передать параметр с именем из Yii::app()->request->csrfParam и значение из Yii::$app->request->getCsrfToken().

Кое что еще.

Давайте посмотрим на некоторые дополнительные функции.

Отключение csrf-токенов для всех действий

1. Если у вас возникли проблемы с enableCsrfvalidation вы можете отключить его.
2. Чтобы отключить CSRF, добавьте этот код в контроллер:

public function beforeAction($action) {

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

}

Отключение csrf-токенов для определенного действия

public function beforeAction($action) {

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

}

Проверка CSRF для Ajax-вызовов

Когда опция enableCsrfValidation включена в основном макете, добавьте 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},

});

Дополнительно [переименовать]

Если ваше приложение требует очень высокого уровня безопасности, например, системы управления банковскими счетами, могут быть приняты дополнительные меры.

Во-первых, вы можете отключить функцию Запомнить меня с помощью config/main.php, следующим образом:

'components' => [

'user' => [

'enableAutoLogin' => false,

].

],

Обратите внимание, что это не будет работать, если параметр enabledSession имеет значение true.

Затем можно уменьшить время ожидания сеанса следующим образом:

'components' => [

'session' => [

'timeout' => 200,

],..

],

Устанавливает количество секунд, после которой данные будут рассматриваться как мусор и очищены.

Конечно, эти меры сделают пользовательский опыт хуже, но они добавят дополнительный уровень безопасности.

Использование GET и POST правильно

HTTP настаивает на том, чтобы не использовать операции GET, которые изменяют данные или состояние. Придерживаться этого правила-хорошая практика. Оно не предотвратит Все типы CSRF, но оно хотя бы снабдит некоторые впрыски, как <img src=,pointless>

Смотрите так же

Чтобы узнать больше о SQL-инъекциях и работе с базами данных через Yii, обратитесь к следующим URL:

* [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)
* Рецепт Предотвращение *XSS* .