



connecting the people that connect the people

## Facebook IFrame Applications and the Zend Framework

### Tutorial 1: Authentication

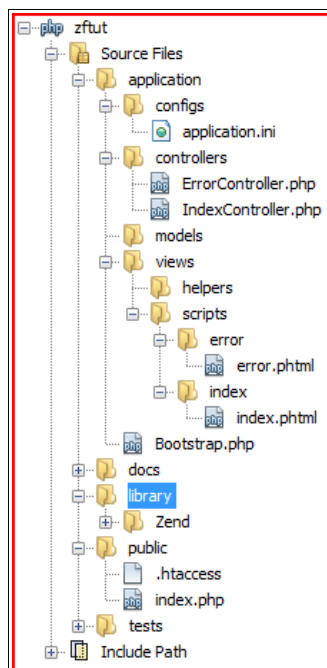
#### Overview

This tutorial is designed to help Facebook and Zend Framework developers authenticate their IFrame applications with the Facebook Platform®. This may not be the BEST method of authenticating a Facebook user, but it is intended as a decent starting point for your own implementation.

#### Getting Started

Let's create a sandbox Zend Framework (ZF) site so that we can get started. I will assume that you have used Zend Framework before, and are familiar with the Zend\_Tool. If not, I can recommend Rob Allen's tutorial which can be found on his blog, at <http://akrabat.com/zft/>.

In this example, I am using Zend Framework 1.11.0 (Rob uses 1.10 but his tutorials work fine with 1.11). I will assume that you have a brand new site setup and accessible to your browser, in the default ZF MVC layout:



Next up, we need to integrate this into a Facebook IFrame application. To do this, we need to become a Facebook Developer. Login to Facebook, and then go to the Developer application, which

is found at <http://www.facebook.com/developers/>, and click on the *Set Up New App* button:



You'll see the application screen, give your App a name and agree to the conditions then hit *Create Application*, you'll be asked to complete some verification, and away you go.

**Note: Only verified Facebook users can become developers, so you'll have to use a phone or credit card verified account to create the App.**

Fill in the nonsense in the *About* tab, and then hit the *Web Site* tab on the left hand side. The *Application ID* and the *Application Secret* on this page are really important; we'll be using them late to authenticate the logged in Facebook user. But for now, simply enter the URL of the the site you setup a new Zend Framework project earlier, and it's domain (if applicable):

A screenshot of the Facebook 'Core Settings' page. It has a light blue background. At the top, there's a header 'Core Settings'. Below it, there are four rows of settings. The first row is 'Application ID' with a blue scribbled-out value and a label 'Your OAuth client\_id'. The second row is 'Application Secret' with a blue scribbled-out value and a label 'Your OAuth client\_secret'. The third row is 'Site URL' with a text input field containing 'zftut.garyhockin.co.uk' and a label 'Your site's URL'. The fourth row is 'Site Domain' with a text input field containing 'garyhockin.co.uk' and a label 'If set, Facebook will enable authentication on all subdomains (e.g., "example.com" will enable \*.example.com)'. At the bottom, there is a blue button labeled 'Save Changes'.

Then we move onto the *Facebook Integration* tab. Here we will tell Facebook which *Canvas Page* we want, and how where they should load our application from. The *Canvas Page* is the `apps.facebook.com/<yournamehere>` URL that you want the outside world to be able to find your application at. I've decided to go for **ghzftut** as it's suitably obscure, I've turned on *Auto-Resize* because I hate scroll-bars, and added my application URL into the *Canvas URL* field. This whole tutorial is about implementing an *IFrame* app, so obviously I have made sure that is selected:

A screenshot of the Facebook 'Canvas' settings page. It has a light blue background. At the top, there's a header 'Canvas'. Below it, there are six rows of settings. The first row is 'Canvas Page' with a text input field containing 'http://apps.facebook.com/ghzftut/'. The second row is 'Canvas URL' with a text input field containing 'zftut.garyhockin.co.uk'. The third row is 'Secure Canvas URL' with an empty text input field. The fourth row is 'Canvas Type' with two radio buttons: 'IFrame' (selected) and 'FBML'. The fifth row is 'IFrame Size' with two radio buttons: 'Show scrollbars' and 'Auto-resize' (selected). The sixth row is 'Bookmark URL' with an empty text input field.

**Note: Contrary to my screenshots, make sure that your URLs end in a / If you want to avoid the extra error I had to correct...**

Next we want the *Advanced* tab. There are 2 options here that are important. The first is the *Sandbox Mode* option. Enabling this means that only people you have added in the *About* tab. This is useful so that you don't spam all your friends with test stream posts from your sandbox application. The second is the *OAuth 2.0 for Canvas* option. This tells Facebook that when they load our *Canvas Page* into their IFrame, we want them to include encrypted authentication information in a Posted *signed\_request* parameter. We will learn more about this later. The rest of the deprecated items can be left alone, they have no relevance to a brand new app.

**Note: Make sure you hit the *Save Changes* button at the bottom of the page, or all of your good work will be lost!**

You should now be able to access your application at the URL `http://apps.facebook.com/<your canvas page>`:



That is the first step complete, and it's a minor victor in itself!

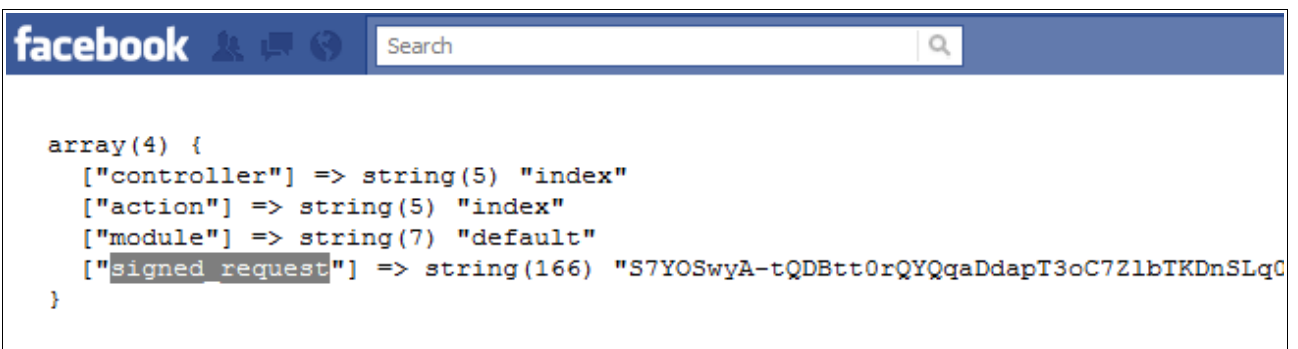
## Decrypting the signed\_request

Next, we need to decrypt the encrypted `signed_request` parameter that Facebook POST to us, let's just check that Facebook are posting us the data first:

```
class IndexController extends Zend_Controller_Action
{
    public function init()
    {
        /* Initialize action controller here */
    }

    public function indexAction()
    {
        Zend_Debug::dump($this->_getAllParams());
        die();
    }
}
```

If you refresh your Facebook application, you should see the parameters passed to the Canvas *Page* through the IFrame dumped to screen:



The screenshot shows the Facebook application interface with a search bar. Below the search bar, a PHP dump of parameters is displayed:

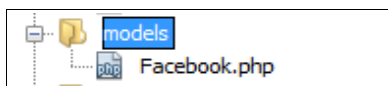
```
array(4) {
  ["controller"] => string(5) "index"
  ["action"] => string(5) "index"
  ["module"] => string(7) "default"
  ["signed_request"] => string(166) "S7YOSwyA-tQDBtt0rQYQqaDdapT3oC7Z1bTKDnSLqC"
```

If you can see the `signed_request` parameter, all is well, otherwise you'll need to check your settings for the application in the Facebook Developers console.

Next, we shall create a model to handle the Facebook authentication. We'll do this using the `Zend_Tool`, so head to your command line, and make sure you are in your project directory, then create the model with:

```
zf create model Facebook
```

You should see the model created in the models directory of your project:



Let's open this up, and add in some basic outline:

```
class Application_Model_Facebook
{
    /**
     * The encrypted Facebook sigs POSTed through signed_request
     * @var string
     */
    private $_fbSigs;
    /**
     * Facebook Application ID
     */
}
```

```

    */
    const FACEBOOK_APP_ID = '<copy application id here>';
    /**
     * Facebook Secret String
     */
    const FACEBOOK_SECRET = '<copy secret here>';

    public function __construct($signed_request)
    {
        $this->_fbSigs = $signed_request;
    }
}

```

We go back to the Facebook Developer console, and we copy our *Application ID* and *Application Secret* into the relevant constants. We then set up a private variable to hold the signed\_request string, and populate that variable in the constructor.

Next, we need to create some private functions that will decode the signed request into its constituent parts.

```

class Application_Model_Facebook
{
    /**
     * The encrypted Facebook sigs POSTed through signed_request
     * @var string
     */
    private $_fbSigs;
    /**
     * Has the user authenticated successfully?
     * @var bool
     */
    public $isAuthenticated = false;
    /**
     * Has the user actually installed (allowed) the app?
     * @var bool
     */
    public $hasInstalled = false;
    /**
     * Facebook passed data
     * @var array
     */
    public $fbData;
    /**
     * Facebook Application ID
     */
    const FACEBOOK_APP_ID = '<copy application id here>';
    /**
     * Facebook Secret String
     */
    const FACEBOOK_SECRET = '<copy secret here>';

    public function __construct($signed_request)
    {
        // Check the signed request is what we would have expected
        if(!is_string($signed_request) || empty($signed_request))
        {
            throw new Exception('Invalid signed_request');
        }
        // Set the private variable to the passed string
    }
}

```

```

$this->_fbSigs = $signed_request;
// Parse the sigs
$this->_parseSignedRequest();
}

private function _parseSignedRequest()
{
    // If the Facebook Sigs are not valid, set authed param and return false
    if (!is_string($this->_fbSigs) || empty($this->_fbSigs))
    {
        $this->isAuthed = false;
        return false;
    }
    // Split the sigs into 2 parts
    list($encoded_sig, $payload) = explode('.', $this->_fbSigs, 2);

    // decode the data
    $sig = $this->_base64UrlDecode($encoded_sig);
    $data = json_decode($this->_base64UrlDecode($payload), true);

    // Check if the encryption is valid
    if (strtoupper($data['algorithm']) !== 'HMAC-SHA256')
    {
        $this->isAuthed = false;
        return false;
    }

    // Check the sigs are valid
    $expected_sig =
        hash_hmac('sha256', $payload, self::FACEBOOK_SECRET, $raw = true);
    if ($sig !== $expected_sig)
    {
        $this->isAuthed = false;
        return false;
    }

    // Sigs are valid, set the variables
    $this->isAuthed = true;
    $this->fbData = $data;

    // If an OAuth token is sent, we know that the user has allowed the app
    if (isset($this->fbData['oauth_token'])
        && is_string($this->fbData['oauth_token'])
        && !empty($this->fbData['oauth_token']))
    {
        $this->hasInstalled = true;
    }

    return true;
}

/**
 * base64 url decoder
 * @param string $input
 * @return string
 */
private function _base64UrlDecode($input)
{
    return base64_decode(strtr($input, '-_', '+/'));
}

```

```
}
```

There are a few points of interest here, the public variable *\$hasInstalled* will be used to store whether the user has allowed the application yet. With Facebook IFrame applications, you only get the logged in user's Facebook ID if the user has allowed your application. We know if the user has allowed the application as the signatures passed do not contain an OAuth token.

Therefore, if the *\$hasInstalled* parameter is false after the model has been instantiated, we need to forward the user to the Facebook canvas allow page:

```
class IndexController extends Zend_Controller_Action
{
    const FACEBOOK_ALLOW_URL = 'https://www.facebook.com/dialog/oauth';

    public function init()
    {
        /* Initialize action controller here */
    }

    public function indexAction()
    {
        // Grab the signed request and setup the Facebook model
        $signed_request = $this->_getParam('signed_request', null);
        if(empty($signed_request))
        {
            throw new Exception('Security Error');
        }
        $FB = new Application_Model_Facebook($signed_request);

        // if the user has not installed,
        // redirect to the allow URL
        if (!$FB->hasInstalled)
        {
            $this->view->redirect_url =
                Application_Model_Facebook::FACEBOOK_ALLOW_URL
                .'?client_id='.Application_Model_Facebook::FACEBOOK_APP_ID
                .'&redirect_uri='.$_SERVER['HTTP_REFERER'];
            $this->_helper->viewRenderer->setScriptAction('redirect');
        }
    }
}
```

In the above code, we check to see if the user has allowed the application, if they haven't we add a *redirect\_url* view variable, and then forward them to the *redirect* view script. Let's create that script now...

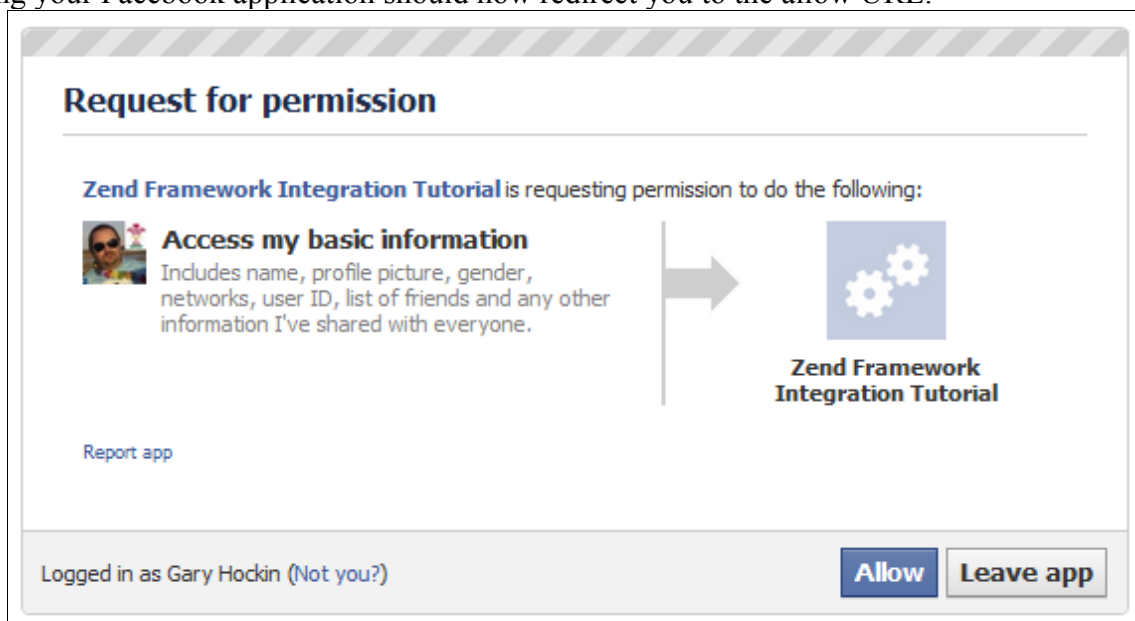
```
zf create view index redirect
```

You should now have a *redirect.phtml* file in your index view scripts folder (found at *application/views/scripts/index/redirect.phtml*). We now want to perform a Javascript redirect in this view script. The reason we cannot do a simple header based redirect in the PHP is because we are stuck inside a Facebook IFrame, this means that when we do a redirect to another Facebook URL, Facebook gets confused momentarily, and displays a holding page before breaking out of the frame. This is rather ugly, personally, I would rather see a very quick redirection message. The redirect view script should look like this:

Redirecting...

```
<script type="text/javascript">
    top.location = "<?php echo $this->redirect_url; ?>";
</script>
```

Hitting your Facebook application should now redirect you to the allow URL:



At this point, it would be possible to ask for more than just the basic permissions, but for the sake of simplicity, all we are asking for is the most basic permissions possible. We will cover the extended permissions in a later tutorial.

Once you click *Allow*, you should see your normal Zend Framework application screen. Lets change it so that our wonderful Facebook application simply displays your user id. To *IndexController/indexAction* add:

```
$this->view->fbid = $FB->fbData['user_id'];
```

This adds the *user\_id* from the decoded Facebook data array to our view. Next, in our view script *views->scripts->index->index.phtml*, modify as follows:

```
<p><?php echo $this->fbid; ?></p>
```

You should then have your Facebook ID output to the screen... congratulations, you have completed your first Facebook application!

## Next Time

In the next tutorial, I will be covering how to use Zend Framework and the authentication mechanism we developed today to query the Facebook Graph API for interesting information.

## Further Reading

The Facebook documentation on authentication:



<http://developers.facebook.com/docs/authentication>

<http://www.socialdevelopers.net>