Integrating Web Services with OAuth and PHP

by Matthew Frost

a php[architect] guide

Table of Contents

Chapter 5

OAuth 1 Implementation

We've covered OAuth 1 in detail; you have a general idea of why it exists and how it works. Now it's time to look at using OAuth 1 in a more practical sense. If you skipped to this chapter and don't have a decent grasp on the basics of HTTP or a basic of understanding of what OAuth 1 was created to do, I would encourage you to read the previous chapters. This chapter includes a signficant amount of example code. I encourage you to have a clear understanding of any code you didn't write before implementing it in your own project.

This chapter will cover implementations in a couple of different categories. We're going to take a look at existing OAuth 1 libraries and how to use them. We will also investigate two framework implementations, namely Zend Framework and Symfony. We will finish with example exercises where we will actually create some calls to well-known OAuth 1 Service Providers. The code here is going to be more than theoretical, it has been tested to work with all these frameworks and services as of the time of this writing.

Existing Libraries

One of the largest problems we come across in the software development industry is the amount of work it takes to verify whether an existing collection of code is trustworthy enough for us to use. Frequently, we would rather put the work towards getting our project done, even if it means writing our own solution instead of using an existing one. The more quickly we

can understand and verify the existing code will fulfill our needs in a secure manner, the more quickly we can implement it and move on to other aspects of our project or application. I present these existing libraries without opinion as components to consider if OAuth is going to be part of your next project.

OAuth PECL Extension

The PHP Extension Community Library (PECL) has an OAuth package which can be used to make OAuth 1 requests. Since it's an extension, it has to be installed, compiled, and the extension has to be enabled in your `php.ini` file before you can use it to make OAuth 1 requests.

Installing

Installing the PECL extension is pretty straight forward. You'll need to have the PHP source files, build tools (autoconf, automake, libtool, and more), and a compiler. For complete instructions, see the PECL installation docs[1]. Assuming you have everything needed to compile them, at the command line you can type:

```
pecl install oauth
```

An easier alternative, you should be able to install it via your Linux distribution's package repository. For OS X, see Rob Allen's post on setting up PHP & Mysql for specific instructions.[2] It's important to make sure you have permission to install extensions; you may have to run this command as `sudo` or have an admin install the OAuth extension for you.

Once you have the extension installed, you must enable the extension in your `php.ini`. The same permission requirements exist, so you'll either have to use `sudo` or have a server admin edit the file for you. Once you've located your `php.ini` (which you can do with php -i from the command line), all you need to do is add the line:

```
extension=oauth.so
```

If you are writing a command line script, it should start working. If you are running a script through a web server, restart your web server to enable the extension. You can check if it's installed with the function `phpinfo()`. You should see the output as in Fiugre 5.1.

FIGURE 5.1

OAuth

OAuth support	enabled
PLAINTEXT support	enabled
RSA-SHA1 support	enabled
HMAC-SHA1 support	enabled
Request engine support	php_streams, curl
source version	$Id: oauth.c 325799 2012-05-24 21:07:51Z jawed $
version	1.2.3

[1] http://php.net/install.pecl.intro

[2] *Setting up PHP & MySQL on OS X Mavericks:* http://akrabat.com/phpmavericks

Code

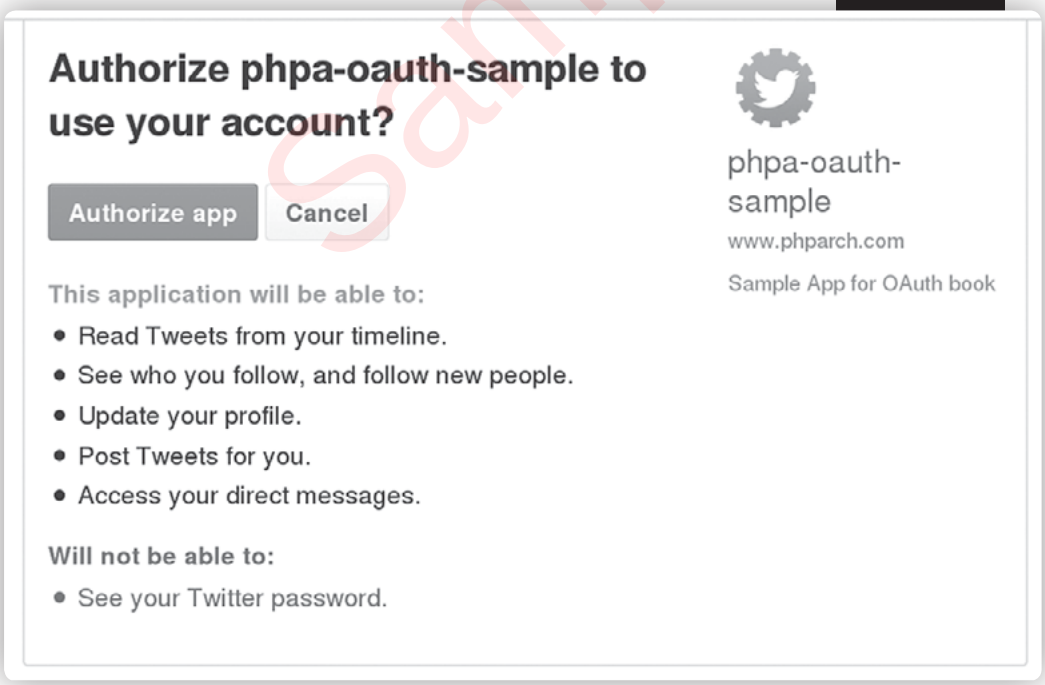
In this section, we're actually going to review code. The first code example demonstrates how to retrieve consumer tokens. The other snippet of code uses the PECL extension to retrieve a tweet from the Twitter REST API. In addition to the code, there will be an explanation as to what the code is actually doing. It is important to note this code snippet won't utilize every available option, but it should give you a good idea of how to use the extension.

When you create and register an application on Twitter, for example, you are assigned an API key and a handful of URLs which will ensure a user gets authenticated correctly.

> *NOTE: Head over to https://apps.twitter.com to register a Twitter app. Once you register, you will find your API key under* **Application Settings**. *Ensure that under the* **Permissions** *tab you ask for Read, Write, and Access Direct Messages access.*

Since we're asking Twitter (in other examples it could be some other service) to handle the difficult part of ensuring a user is who they say they are, we have to be prepared to receive the response from the authentication request. The application itself has an API Key and an API Secret, which allows us to identify the context in which we are trying to use the API. A user has the same type of tokens, which will allow Twitter to identify them and make sure they are interacting with the API in a way allowed by the application. Let's take a look at how the PECL extension allows us to do this.

FIGURE 5.2



Use PHP's built in web server to try the code below. Save Listing 5.1 as `index.php` *and Listing 5.2 as* `callback.php` *in an empty directory. Then start it with* `php -S 127.0.0.1:8080` *while in that directory. For this to work, you'll have to set* `http://127.0.0.1:8080/callback.php` *in your Twitter application as the callback URL.*

Listing 5.1. index.php

```
01. <?php
02. /**
03.  * This code is going to show us how we retrieve OAuth tokens with the
04.  * PECL OAuth Extension
05.  */
06.
07. // API Key/Secret keys
08. $api_key = 'YOUR_APPLICATION_KEY';
09. $api_secret = 'YOUR_APPLICATION_SECRET';
10.
11. // the urls we'll need to authorize/authenticate
12. $base = 'https://api.twitter.com';
13. $request_url = $base . '/oauth/request_token';
14. $access_url = $base . '/oauth/access_token';
15. $authorize_url = $base . '/oauth/authorize';
16.
17. try {
18.     $oauth = new OAuth($api_key, $api_secret, OAUTH_SIG_METHOD_HMACSHA1,
19.                        OAUTH_AUTH_TYPE_URI);
20.     $oauth->enableDebug(); // turn this off in production...
21.
22.     // First we need to get our temporary OAuth
23.     // credentials (Request Tokens)
24.     $request_token = $oauth->getRequestToken($request_url);
25.
26.     // this will send info back that we can use to authorize
27.     header('Location: ' . $authorize_url . '?oauth_token='
28.            . $request_token['oauth_token']);
29. } catch (OAuthException $e) {
30.     print_r($e);
31. }
```

Discover the php[**architect**] guide Book Series

The php[**architect**] series of books cover topics crossing all aspects of modern web development. We offer our books in both print and digital formats. Print copy price includes free shipping to the US. Books sold digitally are available to you DRM-free in PDF, ePub, or Mobi formats for viewing on any device that supports these.

Explore more at http://www.phparch.com/books/