Opauth Documentation

Release 1.0.0-dev

U-Zyn Chua, Marc Ypes

February 21, 2016

Contents

1 Getting started **3**
- 1.1 Configuration . . . 3
- 1.2 Simple example . . . 4
- 1.3 Advanced examples . . . 4

2 Opauth configurations **7**
- 2.1 HTTP clients . . . 7
- 2.2 Strategy configuration . . . 8

3 Response **9**
- 3.1 Response properties . . . 9

4 Available strategies **11**
- 4.1 Create a strategy . . . 11

5 Extend Opauth **13**
- 5.1 Extend Opauth core . . . 13
- 5.2 Strategy contribution guide . . . 13

6 Improvements of v1 **15**

7 Migration guide **17**
- 7.1 Migrating your application . . . 17
- 7.2 Migrating strategies . . . 18

8 Contribute **21**

9 Support **23**

10 Changelog **25**
- 10.1 1.x . . . 25
- 10.2 0.x . . . 25

11 Indices and tables **27**

Opauth is a multi-provider authentication framework for PHP, inspired by OmniAuth for Ruby

Opauth enables PHP applications to do *user authentication* with ease, by providing a standardized method for PHP applications to interface with authentication providers.

For many authentication providers we have strategies. A strategy is the adapter specific to a certain authentication provider. If Opauth does not have a strategy for your favorite provider, it's easy to *create* one.

Contents:

CHAPTER 1

Getting started

The easiest way to start with Opauth is using composer. Choosing to do a manual installation of Opauth will require additional code for autoloading in your application.

We will use the Facebook strategy in the following example, but you can use another strategy as well to get started. Just make sure you check the strategy README for the correct strategy configuration keys. Also the strategy README files will include more info about setting up everything specific to the provider.

You can add Opauth and strategies to your applications `composer.json`:

```
{
    "require": {
        "opauth/opauth": "~1.0",
        "opauth/facebook": "~1.0"
    }
}
```

Note: While Opauth and the strategies have not reached `stable` your root `composer.json` also needs to have:

```
"minimum-stability": "dev"
```

Next you need to run:

```
$ composer install
```

Alternatively you can use the command line:

```
$ composer require opauth/opauth:~1.0
$ composer require opauth/facebook:~1.0
```

This will add Opauth and Facebook strategy to your applications `composer.json` and install them immediately.

1.1 Configuration

Next you to define your configuration. How/where you define and load the configuration array is entirely up to you. The easiest way to configure Opauth is to use a single array that contains the both the configs for Opauth itself and the strategies. Opauth config values are at the root level of the array, while the strategy configurations are stored in the `Strategy` key. The array would look something like:

```
$config = array(
    'Strategy' => array(
        'Facebook' => array(
            'app_id' => 'your_key',
            'app_secret' => 'your_secret'
        ),
    ),
    'path' => '/opauth/'
);
```

Check the configurations section for all possible configs.

1.2 Simple example

Next we will create `opauth.php` with the following contents:

```
<?php
require 'vendor/autoload.php';
$config = array(
    'Strategy' => array(
        'Facebook' => array(
            'app_id' => 'your_key',
            'app_secret' => 'your_secret'
        ),
    ),
    'path' => '/opauth.php/'
);
$Opauth = new Opauth\Opauth\Opauth($config);
try {
    $response = $Opauth->run();
    echo "Authed as " . $response->name . " with uid" . $response->uid;
} catch (OpauthException $e) {
    echo "Authentication error: " . $e->getMessage();
}
```

Set `DocumentRoot` of your web server to this directory, or create a vhost as this example does not work when opauth is in a subdirectory.

Now point the browser to `http://localhost/opauth.php/facebook` to see it in action.

1.3 Advanced examples

Opauth v1 is more flexible then the 0.4 series, meaning you can use your own request parser class and inject strategies manually. If you want to handle the request parsing yourself, you can create a class for this, which must implement `Opauth\Opauth\ParserInterface`

You can now inject your own parser into Opauth's constructor:

```
<?php
use Opauth\Opauth\ParserInterface;

class MyParser implements ParserInterface
{

    public function __construct($path = '/')
    {
        //your implementation
    }

    public function action()
    {
        //your implementation
    }

    public function urlname()
    {
        //your implementation
    }

    public function providerUrl()
    {
        //your implementation
    }
}

//Inject your parser object into Opauth constructor
$Opauth = new Opauth\Opauth\Opauth($config, new MyParser('opauth-path'));
$Opauth->run();
```

You can also set a strategy manually, instead of letting Opauth decide which strategy to run based off the parsed request:

```
$Opauth = new Opauth\Opauth\Opauth();
$Opauth->setStrategy(new Opauth\Facebook\Strategy\Facebook($config['Strategy']['Facebook']));
$Opauth->request();
//or
$Opauth->callback();
```

As you can see in the above example, we are not calling `run()` method here, but manually call `request()` or `callback()` methods on Opauth.

CHAPTER 2

Opauth configurations

Instantiation of Opauth class accepts a configuration array as input.

```php
require 'vendor/autoload.php';
$config = array(
    'path' => '/auth/',
    'http_client' => "Opauth\\Opauth\\HttpClient\\Curl",
    'callback' => 'callback',
    'Strategy' => array(
        //strategy configurations should go here
        //See Strategy configuration section
    )
)
$Opauth = new Opauth\Opauth\Opauth($config);
$response = $Opauth->run();
```

- **`path`**
  - Default: `/`
  - Path where Opauth is accessed.
  - Begins and ends with `/`
  - For example, if Opauth is reached at `http://example.org/auth/`, `path` should be set to `/auth/`; if Opauth is reached at `http://auth.example.org/`, `path` should be set to `/`
- **`http_client`**
  - Default: `Opauth\\Opauth\\HttpClient\\Curl` for cURL (requires `php_curl`)
  - Client to be used by Opauth for making HTTP calls to authentication providers.
  - Opauth also ships with other *HTTP clients*.
- **`callback`**
  - Default: `callback`
  - This forms the final section of the callback URL from authentication provider, ie. `http://example.org/auth/strategy/callback`

2.1 HTTP clients

cURL

- Uses cURL for making of HTTP calls.
- Requires `php_curl`
- Default client. Zero configuration needed.

File

- Uses `file_get_contents()` for making of HTTP calls.
- Requires allow_url_fopen to be enabled.
- To use, set `http_client` to `Opauth\\Opauth\\HttpClient\\File`

Guzzle version 4

- Uses latest stable version of Guzzle for making HTTP calls.
- Recommended HTTP client for Opauth
- Not set as default for Opauth due to minimum PHP requirement being >= 5.4.2.
- To use: 1. Composer require `guzzlehttp/guzzle:~4.0` 1. set `http_client` to `Opauth\\Opauth\\HttpClient\\Guzzle`

Guzzle version 3

- Uses Guzzle version 3 for making HTTP calls.
- To use: 1. Composer require `guzzle/guzzle:~3.7` 1. set `http_client` to `Opauth\\Opauth\\HttpClient\\Guzzle3`

Opauth HTTP client is extensible. You can author your own desired clients if you wish.

2.2 Strategy configuration

Each strategy has its own configuration keys. Check the strategy README file for more information. The strategies should be configured in the `'Strategy'` key in the config array, each under its own key that matches the classname of the strategy.

More info...

CHAPTER 3

Response

Opauth now returns a `Response` object, which stores the result of a successful authentication. A Response object must have five properties accesible publicly: *provider*, *raw*, *uid*, *name*, and *credentials*.

3.1 Response properties

- *provider* - The provider with which the user authenticated (e.g. 'Twitter' or 'Facebook')
- *raw* - An array of all information gather about a user returned by the provider.
- *uid* - A user identifier unique to the given provider, such as a Twitter user ID.
- *name* - A user name unique to the given provider, such as a Facebook username.
- *credentials* - If the authenticating service provides some kind of access token or other credentials upon authentication, these are passed through here.
- *info* - An array containing information about the user, such as name, image, location, etc.

CHAPTER 4

Available strategies

The current version of Opauth has strategies for the following providers

Provider	Maintainer	Composer require
Facebook	Opauth	opauth/facebook
Twitter	Opauth	opauth/twitter
Google	Opauth	opauth/google
GitHub	Opauth	opauth/github
LinkedIn	Opauth	opauth/linkedin
Live	Opauth	opauth/live
Elance	augusto-cdxs	opauth/elance

4.1 Create a strategy

If there is no strategy listed for your favorite provider, it's easy to create one yourself.

More info soon..

CHAPTER 5

Extend Opauth

5.1 Extend Opauth core

The following components of Opauth v1 is fully extensible:

- **Request parser**
  - Opauth makes decision on which strategy, method, action to call based on URL.
  - You can override this if you wish to make a different decision.
  - Refer to `Request/Parser.php` and `ParserInterface.php` for more details.
- **HTTP Client**
  - Opauth uses by default cURL for making http requests and has some other built-in clients.
  - Your own http client can be created if none of the built-in ones fits your needs.
  - Refer to `HttpClientInterface.php` for more details if you wish to create your own HTTP clients.
- **Strategy**
  - Strategies are specific instructions for Opauth on how to handle 3rd-party provider's authentication steps, which can be hugely different from one to other.
  - See Available strategies for known list of Opauth-managed and community-contributed strategies.
  - Or see the Strategy contribution guide below if you would like to author your own.
- **Omit Opauth class**
  - You can even choose not to use the `Opauth` class and just use the Strategies directly if needed, although this will require additional code in your application. This possibility has been made possible based on users requesting for this in the 0.4 cycle. In most cases you can probably still use `Opauth` class now and use a custom request parser class to bend it to your needs.

5.2 Strategy contribution guide

Before *writing your own strategy*, you might want to check out the Available strategies list to see if it is already created, or if the existing one fits your requirements.

To start, you might want to refer to either of the following strategies as guide:

- Twitter for oAuth 1-based strategies

- Google for oAuth 2-based strategies

CHAPTER 6

Improvements of v1

The new release of Opauth has brought about the following changes and improvements as compared to the beta releases:

- Cleaner code base and API
- PSR-1, PSR-2 and PSR-4 compliance

  Opauth is now fully compliant with the following PSR's by PHP Framework Interop Group (PHP-FIG):

  - PSR-1
  - PSR-2
  - PSR-4
- Extensible components

  Opauth is now more extensible than ever. Do not like how our parser works? You can easily extend or override it. The same can be said for many other components on Opauth. See Extend Opauth.
- More streamlined callbacks

  Opauth no longer does another internal callback to pass data back to your app. Now it simply returns the response. With this change, security components and v0.x transport mechanisms have been dropped, as they are no longer needed.
- PHP >= 5.3

  With the use of namespace, Opauth 1.0 is dropping support for PHP 5.2 and supports PHP >= 5.3.
- Tighter integration with Composer

  Opauth now makes full use of Composer for loading of strategies and any related dependencies.
- Response object

  Opauth now returns a more flexible and consistent Response object.
- Moving beyond personal project

  Opauth welcomes Ceeram to the core team. With this addition, Opauth is no longer a personal project of U-Zyn Chua alone but an organization. Check out our new web page.

CHAPTER 7

Migration guide

7.1 Migrating your application

To upgrade your application (or framework specific plugin) from v0.4 to v1.0 you need to use Composer and go through the following changes:

- Update the `composer.json` file of your application, if you had any, else you need to create the file in the root of your application. The `composer.json` should look something like:

```
"require": {
    "opauth/opauth": "~1.0",
    "opauth/facebook": "~1.0",
    "opauth/twitter": "~1.0"
},
```

  You need to point the versions for Opauth and the strategies you use to the 1.0 series.

  Note: While Opauth and the strategies have not reached `stable` your root `composer.json` also needs to have:

```
"minimum-stability": "dev"
```

- Run the following command: `composer update`, to get the correct versions installed into the `vendor` directory.
- Add `require 'vendor/autoload.php';` in your application to get composers autoloading, if you don't already have this. Make sure you have a recent composer version, which includes PSR4 support. If you run into errors, run: `composer self-update`.
- You can keep the existing Opauth configuration array that you were using in v0.4, although many configuration options have been removed. Please check the configurations section to see the current options.
- In the file where you create an Opauth instance, add the following line at the top:

```
use Opauth\Opauth\Opauth;
```

- Update your code to use the new Response object
- Benefit!

7.2 Migrating strategies

For this example we show how to convert `class ExampleStrategy` from v0.4 to v1.0

To upgrade existing strategies to Opauth v1 you need to take the following steps:

- Update `require` and `autoload` in the strategy `composer.json` file:

```
"require": {
    "php": ">=5.3.0",
    "opauth/opauth": "~1.0",
},
"autoload": {
    "psr-4": {
        "Opauth\\Example\\Strategy\\": "src"
    }
}
```

Note: While Opauth and the strategies have not reached `stable` your root `composer.json` also needs to have:

```
"minimum-stability": "dev"
```

- Create `src/` directory in the root of the project
- Move `ExampleStrategy.php` to `src/Example.php`
- Change the class declaration from:

```
class ExampleStrategy extends OpauthStrategy {
```

to:

```
class Example extends AbstractStrategy {
```

If you would choose not to extend AbstractStrategy, your strategy MUST implement StrategyInterface:

```
class Example implements StrategyInterface {
```

- Add the following lines on the top of `Example.php`:

```
namespace Opauth\Example\Strategy;

use Opauth\Opauth\AbstractStrategy;
```

- If your strategy overrides the constructor, you need to modify its signature to:

```
public function __construct($config, $callbackUrl, HttpClientInterface $client)
{
    parent::__construct($config, $callbackUrl, $client);
}
```

- Next you need to make sure your strategy has both `request()` and `callback()` methods.

  The `request()` method handles the initial authentication request and MUST redirect or throw an `OpauthException`. To redirect you can use `AbstractStrategy::redirect($url, $data = array(), $exit = true)`.

  The `callback()` method handles the callback from the provider and MUST return a `Response` object or throw `OpauthException`.

For error handling `AbstractStrategy` has a convenience method `error($message, $code, $raw = null)` which will throw the exception.

The `AbstractStrategy` also has a convenience method `response($raw)` for returning response objects.

- If your strategy needs to read/write session data, please use the `AbstractStrategy::sessionData($data = null)` getter/setter method.
- To obtain the callback url you can use `AbstractStrategy::callbackUrl()`
- `Response` attributes `$uid`, `$name` and `$credentials` MUST be set.

  You can do this either using the response map:

```
//in your ``callback()`` method
$response = $this->response($credentials);
$responseMap = array(
    'uid' => 'id',
    'name' => 'name',
    'info.name' => 'name',
    'info.nickname' => 'screen_name'
);
$response->setMap($responseMap);
return $response;
```

  or directly assiging values to the attributes themselves:

```
//in your ``callback()`` method
$response->credentials = array(
    'token' => $results['oauth_token'],
    'secret' => $results['oauth_token_secret']
);
return $response;
```

  Opauth will use the response map to set values from the raw response to the `Response` class attributes. This replaces the multiple calls to `OpauthStrategy::mapProfile($person, 'username._content', 'info.nickname');` in version 0.4.

  The argument for `AbstractStrategy::setMap($map)` should be an array, with keys pointing to dotnotated paths to the `Response` attribute names and values containing the path to the raw data value.

- If your strategy uses tmhOauth library, please add it as composer required library, instead of adding it as gitmodule or including the code itself.

For more information about creating 1.0 strategies please check the *Create a strategy* section

Now that you are done migrating your strategy we would like to ask you to take the following into account:

- Opauth itself now uses PSR2 coding standards. It is recommended to choose a coding standard for your strategy. Ofcourse you are free not to use this or any other standard. Please at least mention which standard to be used, if any. You can easily check if your strategy matches your standard with php-codesniffer.

  Just run from commandline: `phpcs --standard=PSR2 --extensions=php src/` and fix any errors/warnings if there are any.

  Using a standard helps readabilty for other developers to contribute.

- Please submit your strategy to packagist if you haven't already. The package name would be the Opauth vendorname and your strategyname, divided by a forward slash. The above example would result in `opauth/example`. Once its added to packagist we can add your strategy to the list of supported strategies for version 1.0. Ofcourse you are free to use your own vendorname instead of Opauth's, but using opauth will make it more easy to be found.

If you need help with upgrading or you have other questions, please contact us for support

CHAPTER 8

Contribute

As with any open source project all help is very welcome. You can help by submitting pull requests or issues at github. Also documentation changes can be done by github pull requests. Furthermore you can *create* your own strategy if your favorite provider isn't listed yet and tell us about it, so we can add it to the list.

If you want to have your offline documentation, cd to the docs directory and run:

```
make html
```

Next you can open up `docs/_build/html/index.html` in your browser to view this documentation locally.

Also blogging, tweeting or promoting Opauth in any other way will help the project.

CHAPTER 9

Support

There are several ways to get support for Opauth:

- Discussion group: Google Groups

  Feel free to post any questions to the discussion group.
- Issues: Github Issues
- Twitter: @uzyn
- Email me: chua@uzyn.com
- IRC: #opauth on Freenode

CHAPTER 10

Changelog

10.1 1.x

1.0.0-alpha.1

- Scheduled for release in April, 2014.
- This release is not backward-compatible with v0.x strategies or consumer plugins.
- Initial v1.x release.

10.2 0.x

0.4.4

- Released on May 10, 2013.
- Added HTTP User-Agent header.

0.4.3

- Released on January 10, 2013.
- Fixed a `serverPost()` bug where user-supplied options were not applied correctly.

0.4.2

- Released on August 28, 2012.
- Fix session to check for `session_id()` instead of `$_SESSION`.

0.4.1

- Released on July 22, 2012.
- Not starting session if session is already started.
- Fixed incorrect error message.
- Removed `@` for `file_get_contents`.

0.4.0

- Released on 10 June 2012.
- `mapProfile()` and `clientGet()` for OpauthStrategy class.

0.3.0

- Released on May 30, 2012.
- Introduced unit testing.
- More consistent naming of Strategy's internal properties.
- Smarter loading of strategy, able to make a few guesses on where the class file might be at.

0.2.0

- Released on May 23, 2012.
- Opauth is now Composer compatible and listed on Packagist.
- Opauth now supports autoloaders.
- If a strategy is not autoloaded, Opauth falls back and searches for it at `strategy_dir` defined in config.
- Class name for strategy Foo should now be FooStrategy instead of Foo.
- This is to reduce the likelihood of class name collision due to Opauth not requiring the use of namespace.
- v0.1.0-type class name, ie. Foo, still works, but is now deprecated.

0.1.0

- Released on May 22, 2012.
- Initial release

CHAPTER 11

Indices and tables

- genindex
- search