Zend_Oauth - Pádraic Brady

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Oauth Component Proposal

Proposed Component Name	Zend_Oauth
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Oauth
Proposers	Pádraic Brady Matthew Weier O'Phinney, Zend liaison
Revision	1.0.1 - 1 July 2008 (wiki revision: 16)

Table of Contents

1. Overview
Future Developments
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

The OAuth protocol was published in its final Specification 1.0 on 4 December 2007. It is a protocol allowing websites, web applications or desktop applications to access Service Resources via an API without requiring Users to disclose their credentials. It is an open and decentralised protocol.

A simple use case would be Twitter. At present, Twitter applications such as Spaz.air or Twitterer usually require a User's login username and password (their credentials) in order to access the timeline of other Users they are following or send updates (tweets). This raises a risk that such applications may use those credentials to change the User's password, send "tweets" without their permission, or other unauthorised actions allowable through authenticating with a username/password.

Implementing OAuth, such an application would be able to perform limited authorised actions without requiring Users to disclose their credentials. In effect, this is similar to establishing an API Key and indeed OAuth builds upon existing standards. But the net effect is one of limited access, with OAuth external applications are given defined limited authorisation which can be limited according to function, resource or timeframe.

OAuth is therefore perfect also in situations where a Service Provider is not aware of a User's credentials, as is the case when a Provider implements OpenID. In OpenID, credentials are centralised to a single OpenID Provider and implementing Consumers will require an alternate means of allowing authenticated Users to access their Service Resources via an API. OAuth is not an OpenID extension, but does complement it.

The implementation of OAuth is quite flexible, and the specification is marked "Core" to highlight the ability of Service Providers to create extensions and utilise more secure or different means of exchanging messages.

Future Developments

It's worth noting that OAuth is not a static protocol. Discussions for an OAuth Core 1.1 Specification have been underway for some time and recently entered public comment stage. These changes will be rolled into future Zend_Oauth iterations as a matter of course. The expected changes include:

1. Adoption of standard error codes and human readable error messages in responses (required for simple interoperability across all Service Providers)
2. Adoption of a Service Provider Discovery protocol similar to OpenID (i.e. XRDS-Simple and Yadis)
3. Inclusion of language determination
4. Clarification of scaling and security concerns

In addition the Core 1.0 Specification may be extended through the use of Extensions. Several draft Extenstion Specs are already doing the rounds:

1. OAuth Session Extension Draft 0.1
2. OAuth Discovery 1.0 Draft 2

Also, while waiting for Zend_Service_Yadis approval there is already need for a major revision and updated testing suite to support new specification drafts:

- XRDS-Simple 1.0 (Draft 1)
- Extensible Resource Identifier (XRI) Resolution Version 2.0 (Draft 3)

2. References

- OAuth Website
- OAuth Core 1.0 Final Specification
- Paddy's Offsite Subversion Repository

See above for copies of related Specifications and Draft Specifications.

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** implement the entirety of the OAuth Core 1.0 Final specification without exceptions
- This component **will** implement a fully featured Consumer
- This component **will** implement a fully featured Provider/Server
- This component **will** implement the most recent draft of the OAuth Discovery 1.0 Draft Specification

4. Dependencies on Other Framework Components

Zend_Http_Client
Zend_Crypt_Hmac (Incubator)
Zend_Crypt_Rsa (Incubator)
Zend_Service_Yadis (Incubator)
Zend_Exception
Zend_Uri

5. Theory of Operation

Hypothetical Example:

A Service Provider engaged in allowing User's to answer one question "What are you NOT doing?" has determined that requiring User credentials to access their Web Service API is unnecessarily risky. To improve User security, ease development by adopting an open standard, and allow for future OpenID adoption, they determine to implement OAuth 1.0.

The Service Provider provides to Developers details of how to register for a Consumer Key and Consumer Secret, and declares several new URLs including:

Request Token URL:
https://example.com/request_token, using HTTP POST
User Authorization URL:
http://example.com/authorize, using HTTP Authorize Header
Access Token URL:
https://example.com/access_token, using HTTP POST

The Service Provider provides support for HMAC-SHA1 message signatures. RSA-SHA1 is also a possible alternative (though not for this example!).

1. Obtaining an initial Request Token (Requesting Approval)

Joe Bloggs, a User, visits a third-party web application from which he wishes to view and post new updates to the Service Provider. The application tries to access the User's posts but receives a HTTP 401 Unauthorized header including the following response header:

WWW-Authenticate: OAuth realm="http://example.com/authorize"

The application sends back a HTTP POST request:

```
https://example.com/request_token?oauth_consumer_key=dpf43f3p2l4k3l03
&oauth_signature_method=PLAINTEXT&oauth_signature=kd94hf93k423kf44%26
&oauth_timestamp=1191242090&oauth_nonce=hsu94j3884jdopsl&oauth_version=1.0
```

The Service Provider verifies the oauth_signature value and sends back an unauthorised (requires User authorisation) Request Token in the response body:

```
oauth_token=hh5s93j4hdidpola&oauth_token_secret=hdhd0244k9j7ao03
```

2. Obtaining User Authorisation of Request (Approve Request)

The application at this point redirects the User to the Service Provider Authorization URL so that Joe can approve the application's access to the Service Provider.

```
http://example.com/authorize?oauth_token=hh5s93j4hdidpola
&oauth_callback=http%3A%2f%2fprinter.example.com%2Frequest_token_ready
```

Joe will log in using either his credentials, or via OpenID. He may then approve the application's access, set limits, and perhaps even time the access to a specific timeframe. Once approved, the Service Provider will redirect back to the Consuming web application with:

```
http://example.net/request_token_ready?oauth_token=hh5s93j4hdidpola
```

3. Obtaining an Access Token on foot of an approved Request Token (Gaining Access)

Now that the web application is approved for access, they can request an Access Token to replace the Request Token:

```
https://example.com/access_token?oauth_consumer_key=dpf43f3p2l4k3l03
&oauth_token=hh5s93j4hdidpola&oauth_signature_method=PLAINTEXT
&oauth_signature=kd94hf93k423kf44%26hdhd0244k9j7ao03&oauth_timestamp=1191242092
&oauth_nonce=dji430splmx33448&oauth_version=1.0
```

The Service Provider will check the signature, and reply with an Access Token in the response body:

```
oauth_token=nnch734d00sl2jdk&oauth_token_secret=pfkkdhi9sl3r4s00
```

The Consuming application is now ready to make authorised requests of the Web Service API. If the Web Service API URL is unsecured (not HTTPS) then subsequent requests must be signed using HMAC-SHA1 to protect the Access Token from being intercepted and being used by an unauthorised party.

Security Notes: PLAINTEXT style requests are intended only for use across a secured connection. Otherwise a cryptographic method such as Diffie-Hellman may be utilised (unspecified by OAuth 1.0 of course). Consumer Access Tokens should not be used to assume a Consumer is a User. Access Tokens should allow only limited authorised actions. Phishing remains an issue. Users must be careful that Service Provider URLs

they are redirected to for entering credentials are authenticated.

6. Milestones / Tasks

- Milestone 1: Write unit tests capturing the agreed interface and behaviour
- Milestone 2: Write the code required to pass all unit tests
- Milestone 3: Write integration tests to cover various edge cases
- Milestone 4: Verify that code operates under PHP 5.3 (an Openssl API issue)
- Milestone 5: Complete documentation

7. Class Index

- Zend_Oauth
- Zend_Oauth_Consumer
- Zend_Oauth_Server
- Zend_Oauth_Http
- Zend_Oauth_Http_RequestToken
- Zend_Oauth_Http_AccessToken
- Zend_Oauth_Http_UserAuthorisation
- Zend_Oauth_Http_Utility
- Zend_Oauth_Client
- Zend_Oauth_Token
- Zend_Oauth_Token_Request
- Zend_Oauth_Token_Access
- Zend_Oauth_Token_Access
- Zend_Oauth_Exception

Others may be determined during development

8. Use Cases

Use cases are currently being drafted.

UC-01

A start to finish example using the Ma.gnolia API:

```php
<?php
session_start();

require_once 'Zend/Oauth/Consumer.php';

$options = array(
    'requestScheme' => Zend_Oauth::REQUEST_SCHEME_HEADER,
    'version' => '1.0',
    'signatureMethod' => 'HMAC-SHA1',
    'localUrl' => 'http://your/path/to/this/file.php',
    'requestTokenUrl' => 'http://ma.gnolia.com/oauth/get_request_token',
    'userAuthorisationUrl' => 'http://ma.gnolia.com/oauth/authorize',
    'accessTokenUrl' => 'http://ma.gnolia.com/oauth/get_access_token',
    'consumerKey' => 'YOUR_CONSUMER_KEY',
    'consumerSecret' => 'YOUR_CONSUMER_KEY_SECRET'
);

$consumer = new Zend_Oauth_Consumer($options);

if (!isset($_SESSION['ACCESS_TOKEN'])) {
    if (!empty($_GET)) {
        $token = $consumer->getAccessToken($_GET, unserialize($_SESSION['REQUEST_TOKEN']));
        $_SESSION['ACCESS_TOKEN'] = serialize($token);
    } else {
        $token = $consumer->getRequestToken();
        $_SESSION['REQUEST_TOKEN'] = serialize($token);
        $consumer->redirect();
    }
} else {
    $token = unserialize($_SESSION['ACCESS_TOKEN']);
    $_SESSION['ACCESS_TOKEN'] = null; // forces reset of access token on all example runs ;)
}

// the client is a Zend_Http_Client subclass with built-in OAuth request handling
$client = $token->getHttpClient($options);
$client->setUri('http://ma.gnolia.com/api/rest/2/bookmarks_count'); // we're counting!
$client->setMethod(Zend_Http_Client::POST);
$client->setParameterPost('group','oauth'); // bookmarks for the OAuth Group

$response = $client->request();
header('Content-Type: ' . $response->getHeader('Content-Type'));
echo $response->getBody();

// Retrieved XML will be similar to:

//      <?xml version="1.0" encoding="utf-8" ?>
//      <response status="ok" version="">
//       <count>138</count>
//      </response>
```

UC-02

A start to finish example using the Google Data Contacts API:

```
<?php
session_start();

require_once 'Zend/Oauth/Consumer.php';
require_once 'Zend/Crypt/Rsa/Key/Private.php';

$options = array(
    'requestScheme' => Zend_Oauth::REQUEST_SCHEME_HEADER, // Google does not support POSTBODY option
    'version' => '1.0',
    'signatureMethod' => 'RSA-SHA1',
    'localUrl' => 'http://path/to/this/file.php', // UPDATE FOR YOUR DETAILS
    'requestTokenUrl' => 'https://www.google.com/accounts/OAuthGetRequestToken',
    'userAuthorisationUrl' => 'https://www.google.com/accounts/OAuthAuthorizeToken',
    'accessTokenUrl' => 'https://www.google.com/accounts/OAuthGetAccessToken',
    'consumerKey' => 'exmaple.com',  // UPDATE FOR YOUR DETAILS
    'consumerSecret' => new Zend_Crypt_Rsa_Key_Private(
        file_get_contents(realpath('./myrsakey.pem'))  // UPDATE FOR YOUR DETAILS
    )
);

/**
 * Example utilises the Google Contacts API
 */

$consumer = new Zend_Oauth_Consumer($options);

if (!isset($_SESSION['ACCESS_TOKEN_GOOGLE'])) {
    if (!empty($_GET)) {
        $token = $consumer->getAccessToken($_GET, unserialize($_SESSION['REQUEST_TOKEN_GOOGLE']));
        $_SESSION['ACCESS_TOKEN_GOOGLE'] = serialize($token);
    } else {
        $token = $consumer->getRequestToken(array('scope'=>'http://www.google.com/m8/feeds'));
        $_SESSION['REQUEST_TOKEN_GOOGLE'] = serialize($token);
        $consumer->redirect();
        exit;
    }
} else {
    $token = unserialize($_SESSION['ACCESS_TOKEN_GOOGLE']);
    $_SESSION['ACCESS_TOKEN_GOOGLE'] = null;
}

// OAuth Access Token Retreived; Proceed to query Data API for current user's contacts
$client = $token->getHttpClient($options);
$client->setUri('http://www.google.com/m8/feeds/groups/default/full');
$client->setMethod(Zend_Http_Client::GET);

$response = $client->request();
header('Content-Type: ' . $response->getHeader('Content-Type'));
echo $response->getBody();
```

UC-03

Options may be set using a typical Options array, in the future using Zend_Config, and individually using publicly accessible mutators and accessors:

```
$consumer = new Zend_Oauth_Consumer(
    array(  // options array
        'requestScheme'=>Zend_Oauth::REQUEST_SCHEME_QUERYSTRING
    )
);
$consumer->setVersion('1.0');
$consumer->setRequestMethod(Zend_Oauth::GET);
```

9. Class Skeletons

Please refer to ongoing development in my subversion repository

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>