sanction Documentation

Release 0.1

Demian Brecht

April 22, 2013

CONTENTS

sanction [sangk-shuhn]: authoritative permission or approval, as for an action.

CHAPTER

ONE

OVERVIEW

sanction is a lightweight, dead simple client implementation of the OAuth2 protocol. The major goals of the library are:

- Grok me
  - At a whopping 138 SLOC, sanction is pretty easy to understand
- Support multiple providers
  - Most providers have implemented varying levels of adherance to the OAuth 2.0 spec. The goal with this library is to either handle the diversions natively or expose methods to allow client code to deal with it efficiently and effectively.
- Support all server-side OAuth 2.0 flows
  - Three of the four OAuth 2.0 flows should be supported by this library.

1.1 Conventions

This module follows the following patterns for function calls and callbacks:

1.1.1 Calls

```
libfunc(opts, callback)
```

- `opts`: An object containing all required and optional attributes.
- `callback`: The function to be called on completion of execution.

1.1.2 Callbacks

```
mycallback(e, data)
```

- `e`: An error object. `null` if no errors encountered.
- `data`: The data retrieved by the API call. `null` if error encountered.

CHAPTER

TWO

INSTALLATION

```
npm install [-g] sanction
```

CHAPTER

THREE

USAGE

3.1 Overview

There are three steps to accessing OAuth 2.0-protected resources:

1. Access request/authorization
2. Token exchange (code for access token)
3. Resource request

This library exposes API for all three, but expects your client code to redirect the user as needed for your given environment (web, installed app, etc) for the initial access request and authorization.

3.2 Instantiation

```
var sanction = require('sanction');
var opts = {
    authEndpoint: '[uri]',
    tokenEndpoint: '[uri]',
    resourceEndpoint: '[uri]',
    clientId: '[client_id]',
    clientSecret: '[client_secret]',
    redirectUri: '[redirectUri]'
};
var client = new sanction.Client(opts);
```

- `authEndpoint`: The provider-specific base URL to redirect the user to, to gain authorization to access protected resources.
- `tokenEndpoint`: The provider-specific base URL to use when exchanging the access "code" for a token used in all subsequent resource requests.
- `resourceEndpoint`: The provider-specific base URL to use when executing resource requests.
- `clientId`: The client ID allocated to your app by the provider.
- `clientSecret`: The client secret allocated to your app by the provider.
- `redirectUri`: The redirect URL given to the provider during app registration for security reasons.

3.3 Authorization Redirect

```
var opts = {
    scope: '[scope]',
    state: '[state]',
    responseType: '[response_type]'
};
client.authUri(opts, function(e, uri) {
    // TODO: redirect based on environment
});
```

3.3.1 Required Parameters

- `scope`: A list of provider-specific resources your application is requesting access to.

Note: The delimiter is provider-defined. RFC 6749 specifies space-delimiter, but this is not always the case (i.e. Facebook)

3.3.2 Optional Parameters

- `state`: A string that will be returned to the code handling redirectUri. The intention is to use this as an XSS counter-measure in the form of CSRF protection.
- `responseType`: Defaults to "code". This should only be set if dealing with OAuth 2.0 extensions and you know what you're doing.

3.4 Token Exchange

The token exchange implementation (`requestToken`) deals with two variations of the flow: initial token request (code exchange) and token refresh. The flow followed is determined by the data in `opts`.

```
var opts = {
    code: '[code]',
    parser: [parser]
};

// or

var opts2 = {
    refreshToken: '[refreshToken]',
    parser: [parser]
};

client.requestToken(opts, function(e, data) {
    // TODO: do something with the token if it's there
});
```

3.4.1 Required Parameters

- `code`: The `code` sent back by the provider once authorization has been granted to your application.

- `refreshToken`: The `refresh_token` to use in order to refresh the clients' `access_token`.

Note: `code` *or* `refreshToken` should be defined. A combination or or omission of both will result in an error.

3.4.2 Optional Parameters

- `parser`: Not all providers use JSON. This allows the client code to define what parsing method should be used on provider response.
- `grantType`: Defaults to "authorization_code". Should only be changed if you're dealing with an OAuth 2.0 extension and you know what you're doing.

3.5 Resource Request

```
var opts = {
    path: '[path]',
    method: '[method]',
    data: '[data]',
    parser: [parser],
    transport: [transport]
};
client.request(opts, function(e, data) {
    // TODO: present the data to the user
});
```

3.5.1 Required Parameters

- `path`: Resource path. This is in addition to the base `resourceEndpoint` set during client instantiation.

3.5.2 Optional Parameters

- `method`: HTTP method to be used (`GET`, `POST`, etc). Defaults to `GET` unless `data` is present, in which case `POST` is used as the default.
- `data`: The data payload to be sent along with the request.
- `parser`: As with the token request, this will be used if defined to parse response data.
- `transport`: The access token transport method to use. Defaults to `sanction.transport.query`. The other implementation provided is `sanction.transport.headers`.