OAuth2lib

http://tools.ietf.org/html/ietf-oauth-v2-08 implementation

24 Junio 2010

Índice de contenido

Introduction

OAuth2lib is a library based on OAuth2 that implements the Assertion Profile of the OAuth2 standard.

OAuth2 is the evolution of the OAuth protocol that defines authorization flows in order that a Client application would be able to request resources from a resource Server on behalf of an specific user.

The authorization flows depends on the system in which it's deployed: web applications, desktop applications, mobile phones, and living room devices.

In this library we develop the authorization profile called 'Assertion Profile' in the OAuth2 protocol.

Documentation

OAuth2 Assertion Profile

In order to understand the Assertion Profile, it's necessary to describe the following concepts:

- **OAuth Client:** Application that uses OAuth to access the OAuth Server on behalf of the user or on his own behalf.
- **OAuth Authorization Server:** HTTP server that with an OAuth Client's request is capable of issuing tokens that give access to the resources.
- **OAuth Server:** HTTP Server that has got the protected resources and denies or grants the access to them depending on the token given within the Client's request.
- **Token:**A string representing an access grant issued to the client that has to be delivered to the OAuth Server in order to access to the resource.

OAuth 2 provides a protected resources authorization's delegation to different Trust Authorities.

The Client, in order to access a protected resource, must obtain first an Authorization Server's authorization, in the form of tokens.

This token is obtained by sending some credentials to the Authorization Server.

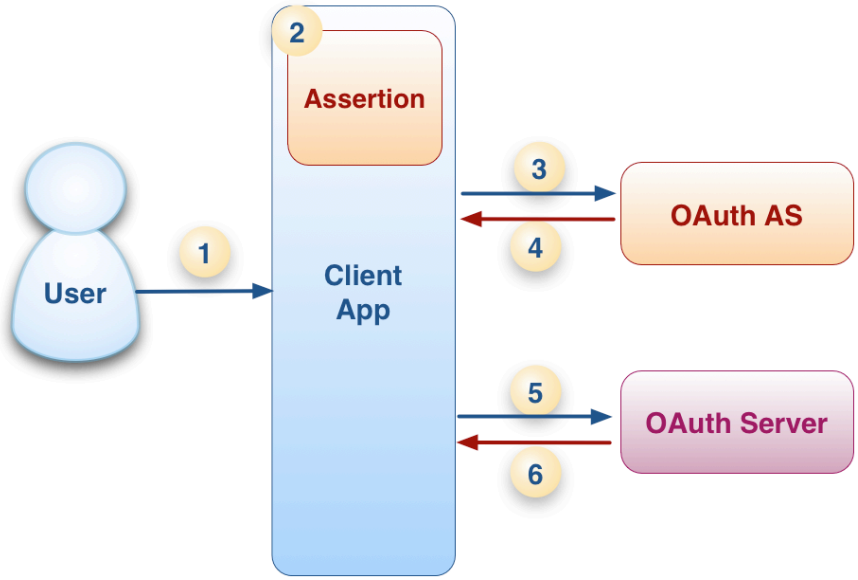
To get the resource, the client will have to give the resource Server the token obtained, and if it's a valid one, the Server returns the resource.

Authorization Flow

In this authorization flow, the peculiarity is that the credentials sent to the Authorization Server are assertions provided by an SP. So far, this library supports SAML2 and PAPI assertions.

The steps taken in order to obtain the protected resource are:

1. The user goes to a Client Application.
2. In the Client App, the user authenticates in an external SP that generates a SAML or PAPI assertion.
3. The Client App sends the assertion obtained to an Authorization Server. There, a token for a certain user, client, scope and lifetime is generated.
4. The Authorization Server sends the generated token to the Client App.
5. The Client App acts on behalf of the user and requests the resource to the Server. This token can be used more times until it expires.
6. The Server returns the resource if the token sent is a valid token.


2
Assertion
3
OAuth AS
4
User
1
Client
App
5
OAuth Server
6

OAuth Client

The OAuth Client Application's goals are the following:

- To request a token from an Authorization Server with the obtained assertion.
- To request the resource on behalf of the user with the token.

In order to achieve the goals of a Client Application, the library provides two classes, called *oauthClient* and *oauthConfig*. See the descriptions of each one below.

oauthClient class

The goal of this class is to interact with the Authorization Server and the resource Server to get a token and a protected resource, respectively.

This class has got two public methods:

requestAccessToken(assertion, scope, url_as, url_rs [,lifetime]):void

This function sends an assertion of one specific user, the scope of the request and other configuration parameters to obtain the token that will provide access to the protected resource.

The parameters required for this method are:

- String assertion: Assertion provided
- String scope: URI of the scope of the resource.
- String url_as: URL of the OAuth authorization server.
- String url_rs: URL of the OAuth resource server.
- String lifetime: [Optional] Lifetime of the access or bearer token. Default value: 3600 seconds.

The obtained token will be stored in a class parameter and it will be used later to get the resource.

requestResource(bearer, method [,extra]):void

Function that makes token requests. If the requests aren't developed HTTP over TLS they will be signed by the methods specificated in erlier versions of the standard.

If the connection is HTTP, clients make token requests by including the access token using the HTTP "Authorization" request header with the "Token" authentication scheme. The access token is included using the "token" parameter. For example, the client makes the following HTTPS request:

```
GET /resource HTTP/1.1 Host: server.example.com
Authorization: Token token="vF9dft4qmT"
```

When the request is HTTP over TLS, it will be used an access token with a matching secret: The client uses the "token" attribute to include the access token in the request, and uses the "nonce", "timestamp", "algorithm", and "signature" attributes to apply the matching secret. For example:

```
GET /resource HTTP/1.1 Host: server.example.com
 Authorization: Token token="vF9dft4qmT",
                    nonce="s8djwd",
                    timestamp="137131200",
                    algorithm="hmac-sha256",
                    signature="wOJIO9A2W5mFwDgiDvZbTSMK/PY="
```

The parameters of this method are the following:

- String bearer: String with the value "bearer" if we want to do a bearer request or "hmac-sha256" if we want to do a cryptographic request.
- String method: Method choosen for the resource request. The available methods are the methods defined by the draft-ietf-oauth-v2-05: URI Query Parameter ("get"), HTTP Authorization request header ("header") and Form-Encoded Body Parameter ("form")
- Array extra:[optional] Array of extra parameters added in case of necessity. Initialized by default to an empty array.

Besides this configuration by parameters, the oauth client class recognizes if the connectios is HTTP or HTTP over TLS and will act in consecuence to it, changing the security of the request if the case may be.

Other public methods

Besides this methods, the *oauthClient* class provide another public methods and functions to improve its use:

- getError():String Return an error if it exists or null if it isn't.
- getAccessToken():String Return the access token or null if it haven't been obtained.
- getExpiresIn():int Return the expiration time of the token in seconds.
- getResource():mixed Return the required resource. The format of the resources can be different in each implementation. The custom transformation of the resource format could be done in the private method 'transformResponse'
- setKey(String key):void Configure the shared secret of the Client
- getError():String Return an error if it exists or null if it isn't.

oauthConfig class

This library offers one additional class, *oauthConfig*, that simplifies the interaction of the application with the OAuth protocol.

This class have a constructor that initializes the parameters required by the Client to start the authorization flow:

- Authorization Server: URL of the RedIRIS's example AS
  - Default value: https://test76.rediris.es/oauth2/oauth_as/tokenEndpoint.php
  - Getter: get_oauth_as():String

    - Setter: set_oauth_as(as)

- Resource Server: URL of the RedIRIS's example Server
    - Default value: https://www.rediris.es/sir/api/oauth_server/serverEndpoint.php
    - Getter: get_oauth_server():String
    - Setter: set_oauth_server(server)

- Scope: Scope of the RedIRIS's example service.
    - Default value: http://www.rediris.es/sir/api/sps_available.php
    - Getter: get_scope():String
    - Setter: set_scope(scope)

- Assertion Type: URN of the assertion's type. Supported by this library: PAPI (urn:sir...) or SAML2 (urn:saml2...)
    - Default value: urn:sir
    - Getter: get_assertion_type():String
    - Setter: set_assertion_type(assertion_type)

- Lifetime: Lifetime of the access or bearer token, in seconds.
    - Default value: 3600
    - Getter: get_lifetime():int
    - Setter: set_lifetime(lifetime)

- Token type: Type of the request of the resource. It can be with an Access Token('hmac-sha256') or an Bearer Token ('bearer').
    - Default value: bearer
    - Getter: get_token_type():String
    - Setter: set_token_type(token_type)

- Format: Format of the response to the Access Token Request. The supported values are the specified by '3.3.2. Response format' of the draft-ietf-oauth-v2-05: application/json, application/xml and application/x-www-form-urlencoded.
    - Default value: application/json
    - Getter: get_response_format():response_format
    - Setter: set_response_format(response_format)

- Key: Shared secret of the Client Application.

- Getter: get_key():String
- Setter: set_key(key)

There is, too, a method that gets the error if something goes wrong. This method is getError and returns null if the flow was finished correctly or with a string with the error message on the contrary.

startFlow(assertion):mixed

In order to start the authorization flow, the *oauthConfig* class provide this public function that makes all steps of the OAuth2 authorization flow.

This method's goal is minimize the code that the developers have to implement to deploy an OAuth Client Application.

The parameter that this method need is the assertion provided by an Service Provider in form of an array. (For example, in phpPoA it would be 'userdata' and in simpleSAMLphp '$as->getAttributes()').

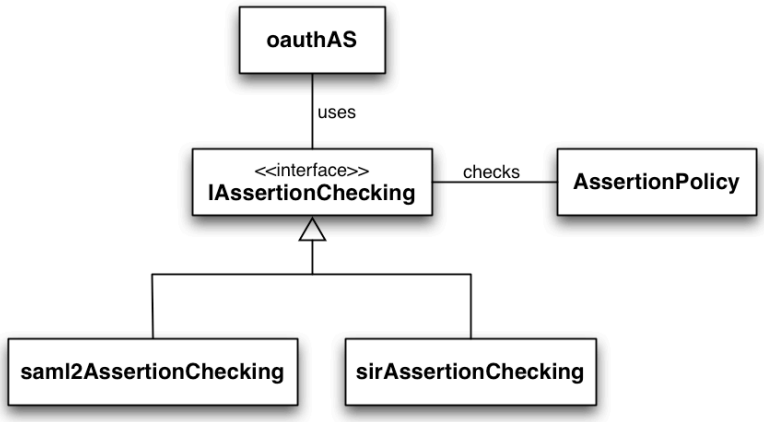
This method returns the resource if everything goes ok or null otherwise.

OAuth Authorization Server

The OAuth Authorization Server's goals are the following:

- Given an assertion, to check if it's a valid one and to generate a token for an specific scope.

In order to achieve the goals of Authorization Server, the library provides the following structure:



- **oauthAS class:** Class that has the logic of the Authorization Server. It generates the token and manage the Client's requests.
- **AssertionPolicy class:** Class that manages the policy.xml file, wich one has got the authorization policy for each type of assertion.
- I**AssertionChecking interface:** Interface that defines the methods for an Assertion Checking class.
- **saml2AssertionChecking:** Class that checks if an assertion in saml2 format fulfill the defined policies.
- **SirAssertionChecking:** Class that checks if an assertion in saml2 format fulfill the defined policies.

See the descriptions of each element below.

oauthAS class

The goal of this class is to interact with the Client to manage a token request and manage the generation and return of the requested token.

This class has got three public methods:

manageRequest(post, hostname):void

Function that manages the request of the Client Application and return an appropiate response, depending on the response_format parameter. Return an HTTP 400 Bad Request with a json_encode error in the body of the response

The parameters required by this method are:

- Array post: An array with the parameters of the request. It contains:

- String type: The parameter value MUST be set to "assertion".
- String format: The format of the assertion as defined by the authorization server. The value MUST be an absolute URI. The types supported by this library are PAPI assertion and SAML2 assertion.
- Array assertion: The assertion.
- String scope: (Optional) The scope of the access request expressed as a list of space-delimited strings. The value of the "scope" parameter is defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope.
- Int lifetime: (Optional) The lifetime of the access token in seconds.

- String hostname: String with the IP of the Client requesting the access token.

getLifetime():int

Method that returns the lifetime of the token.

setLifetime(int):void

Method that sets the lifetime of the token.

IAssertionChecking interface

Interface that defines the public methods that an implementation of it has to develop in order to fulfill the required functionality. This functions are:

checkAssertion(assertion): bool

Method that checks the assertion given as a parameter. Returns true if it's a valid assertion, false if it isn't.

getPersonId(): String

Method that returns the identification of the user owner of the assertion.

getError(): String

Method that returns an error if something goes wrong with the assertion or null if it isn't.

saml2AssertionChecking class

Implementation of the IAssertionChecking that supports a saml2 assertion.

sirAssertionChecking class

Implementation of the IAssertionChecking that supports a PAPI assertion.

AssertionPolicy class

Class that loads a policy from an xml file and checks if the assertion given is a valid one.

Definition of the authorization policy xml file

This xml file define what kind of user can access to the resources.

The main structure looks like this one:

```
<AssertionList>
  <Assertion ...>
                .
                .
                .
  </Assertion>
</AssertionList>
```

Where,

- AssertionList: this element defines the list of assertion types.
- Assertion: this element defines the policies for an specific assertion.

Defining an Assertion Policy

The next example shows how can be defined an authorization policy for an specific assertion.

SAML2 Assertion example

```
<Assertion type="saml2">
    <Policies>
        <Policy>
            <Attributes check="all" >
               <Attribute name="def:eduPersonScopedAffiliation"
value="staff@rediris.es" />
            </Attributes>
        </Policy>
    </Policies>
</Assertion>
```

PAPI Assertion example

```
<Assertion type="saml2">
    <Policies>
        <Policy>
            <Attributes check="all" >
               <Attribute name="ePA" value="staff" />
            </Attributes>
        </Policy>
        <Policy>
            <Attributes check="any" >
               <Attribute name="sHO" value="rediris.es" />
               <Attribute name="sHO" value="fecyt.es" />
```

```
            </Attributes>
        </Policy>
    </Policies>
</Assertion>
```

Where,

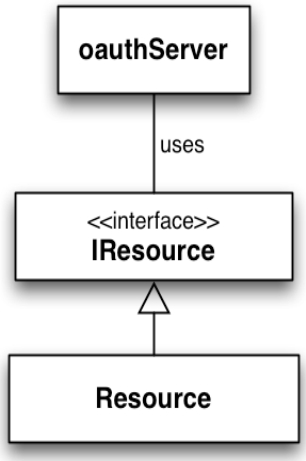
- Inside the element <Policies>, you can add one element <Policy> per policy that you may need.
- The element <Attributes> inside a <Policy> defines how the user's attributes should be checked. This is specified in its attribute check, where allowed values are:
    - all: it says that all the rules have to be satisfied.
    - any: it says that one or more rules have to be satisfied.
    - none: it says that none of all rules has to be satisfied.
- The element <Attribute> inside the one <Attributes> defines a rule over the user's attributes, where name is the name of the attribute and value is its value.

OAuth Resource Server

The OAuth resource Server's goals are the following:

- Given a token, to check if it's a valid one and to return the requested resource.

In order to achieve the goals of Authorization Server, the library provides the following structure:



- **oauthServer class:** Class that has the logic of the resource Server. It checks the token and manage the Client's requests.
- **IResource interface:** Interface that defines the methods for a Resource class.
- **Resource class:** Class that gets the resource if the token is a valid one. Implements the interface IResource.

See the descriptions of each element below.

oauthServer class

The goal of this class is to interact with Client to get a protected resource.

This class has got three public methods:

manageRequest():void

Function that manage the request of the client and give a response with the request if the request has got a valid token and in the case of an error or an invalid token, the response will be an HTTP 400 or 401.

getResource():String

Function that returns the resource

getError():String

Function that gets an error in case of something with the request ist wrong.

IResource interface

Interface that defines the public methods that an implementation of it has to develop in order to fulfill the required functionality. This functions are:

getResource(scope): String

Method that returns the resource defined by the scope parameter and false if it's not possible to return the resource.

Resource class

Class that obtains the resource. It must implements the interface IResource.

This class will be implemented depending on the resources that we are getting.

Installation and configuration

OAuth Client

Installation

In order to install the Client application you just have to include de following archives somewhere accesible for your code:

- oauthClient.php
- Directory utils with:
- bdUtils.php

Configuration

Step1: Instantiate the OAuth configuration class

```
$client = new oauthConfig();
```

Step2: Configure the parameters of the OAuth Client class

In the access point of the Client Application we must configure the parameters of the OAuth Client class. These are defined in the client's documentation and are the following:

- Authorization Server
- Resource Server
- Scope
- Assertion Type
- Lifetime
- Format
- Token type
- Format
- Key

The configuration will be made with the getter and setters of the oauthConfig class, for example:

```
$oauth_as="https://oauth-server.rediris.es/oauth2/oauth_as/tokenEndpoint.php";
$client->set_oauth_as($oauth_as);
$oauth_server="https://oauth-
server.rediris.es/oauth2/oauth_server/serverEndpoint.php";
$client->set_oauth_server($oauth_server);
$key = "123456789abcdefghijklmnopqrstuvwxyz";
$client->set_key($key);
```

Step3: Start the authorization flow

```
$dev = $client->startFlow($assertion);
```

Where $assertion will be the specific assertion for an user. It could be an PAPI assertion or a SAML2 assertion.

Step4: Getting the resource

```
if($dev!=null){
   echo formatResource($dev);
}else{
   echo 'Error: '.$client->getError();
```

}

To show the resources to the user, you must implement a method to visualize them properly.

For example, in the use case example, we've implemented a method that gets the resource, an xml string, and format it properly, to show the information in a readable way.

To know if exists an error, we must check if the result of the startFlow method is null or it isn't.

To know which error has happened, we can use the getError() method, that returns a string with the information.

OAuth Authorization Server

Installation

In order to install the Authorization Server you just have to include de following archives somewhere accessible for your code:

- oauthAS.php
- tokenEndpoint.php
- Directory utils with:
- assertionChecking.php
- sirAC.php
- saml2AC.php
- policy.php
- policies.xml
- Directory keys with:
- keys.xml

Configuration

Step1: Configuring the file keys.xml

We must give the Client Applications a shared secret that will be used to reinforce the security of the application. This shared secret or key will be registered in the keys.xml file. The format of this archive will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<Keys>
<Key id="ip_client" value="key"/>
<Key id="ip_client2" value="Key2"/>
</Keys>
```

Where ip_client will be the ip of the Client Application and key will be the shared secret.

This shared secret has to be the same of the shared secret given for the Resource Server.

Step2: Configuring the file policies.xml

The Authorization Server has to know which assertions are valid ones. To define the authorization policy, we use the policies.xml file. The correct configuration of this archive is defined in the section of the Authorization Server's Documentation.

OAuth resource Server

Installation

In order to install the resource Server you just have to include de following archives somewhere accesible for your code:

- oauthResourceServer.php
- serverEndpoint.php
- Directory utils with:
- bdUtils.php
- Directory keys with:
- keys.xml

Configuration

Step1: Configuring the file keys.xml

We must give the Client Applications a shared secret that will be used to reinforce the security of the application. This shared secret or key will be registered in the keys.xml file. The format of this archive will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<Keys>
<Key id="ip_client" value="key"/>
<Key id="ip_client2" value="Key2"/>
</Keys>
```

Where ip_client will be the ip of the Client Application and key will be the shared secret.

This shared secret has to be the same of the shared secret given for the Authorization Server.