Oauth Document

OmnyPay Payment Platform

Prepared By: Susil Rayaguru

Table of contents

[1 Introduction 3](#_Toc336423703)

[2 Oauth2.0 implementation 5](#_Toc336423710)

1. Introduction

OAuth is a well-known authorization framework that enables applications to obtain authorization access to user accounts on an HTTP service, such as Facebook, GitHub or Yahoo. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. OAuth 2 provides authorization flows for web and desktop applications, and mobile devices. OAuth 2.0 covers different ways a client application can obtain authorization to access the resources stored on the resource server.

OAuth 2.0 defines the following roles of users and applications:

* Resource Owner
* Resource Server
* Client Application
* Authorization Server

|  |
| --- |
| OAuth 2.0 roles as defined in the specification. |
| **OAuth 2.0 roles as defined in the specification.** |

I

The resource owner is the person or application that owns the data that is to be shared. For instance, a user on Facebook or Google could be a resource owner. The resource they own is their data. The resource owner is depicted in the diagram as a person, which is probably the most common situation. The resource owner could also be an application. The OAuth 2.0 specification mentions both possibilities.

The resource server is the server hosting the resources. For instance, Facebook or Google is a resource server (or has a resource server). In our case Omnypay Cloud Server can be thought of as a resource server.

The client application is the application requesting access to the resources stored on the resource server. The resources, which are owned by the resource owner. A client application could be a game requesting access to a user’s Facebook account. In our case Omnypay mobile app can be thought of as a client application.

The authorization server is the server authorizing the client app to access the resources of the resource owner. The authorization server and the resource server can be the same server, but it doesn't have to. The authorization server generates an authorization token (Oauth token) for the authorizing the app for accessing the user resource. This token must be accompanied with the request URL for accessing any authorized resource. Oauth token has got a configurable expire time. Once the token is expired it will no longer be valid. User can refresh the Oauth token to get a new token for continuing access.

### 2 Oauth2.0 Implementation

In omnypay we are implementing Oauth2.0 for user authentication and token generation.

In Omnypay oauth2.0 has been implemented as a two way process.

* Spring Security for authentication.
* Spring Oauth2.0 for authorization and token generation.

Steps for accessing a resource:

1. For accessing any resource the client application need to register with the Oauth Server to get Client id and Client secret key.
2. Client application also need have user id and password for the user whose resource we want to access.
3. The client application should obtain a token from the Oauth server

<http://localhost:8080/mywebservice>/oauth/token? username=user1&password=user1&client\_id=client1&client\_secret=client1&grant\_type=password

For the above steps the Oauth server will verify, the client id and secret for confirming a valid client application, afterwards it will go on authenticating user id and password.

For an authorized user ,it responds with json data

{"access\_token":"aa45e9ac-83a1-4977-898b-43d1eb9933b8","token\_type":"bearer","refresh\_token":"e8208a2c-4b25-425e-a360-23a1b68de315","expires\_in":298825}

Access\_token: should be accompanied while requesting any resource.

Bearer : says anyone who got the token can access resource.

Refresh\_token: refresh\_token is used for getting a fresh token when the token expires.

Expires\_in: specified time after which the token become invalid.

4)

Once the client application got the token he can access resource for the user, using the given token

As   
<http://localhost:8080/mywebservice/profile_page/?access_token=b6a50b54-0cd1-476f-8539-b3b5ffd8f406>

**OAUTH FRAMEWORK [LOGIN SCENARIO]**

Web Se rvices / Cloud Server

Oauth Server

FILTER

CLIENT/MOBLE APP

Client accessing login URL with user credential.

Filter intercepts the client requests and talk to auth server for validation and token generation with client credential

Oauth Server provides response to filter with a token for valid user else with an error message

Oauth Server will access cloud for validating user with client credential credential.

Cloud respond to oauth server for user validation

If token is generated by Oauth Server, filter will create an entry in LoginMap with the token.

Filter respond to client with a token for valid user else respond with an error message.

**Login Flow**

Client application accessing login service with server url (<http://localhost:8080/mobileApp/rest/user/login>) with client credential.

1. Filter will intercept the client request and talk to oauth server with client credential for getting a token (token will only be fetched for an authenticated client)
2. Oauth server will access cloud server for validating the client credential.
3. Cloud server will respond to Oauth server with validation status.
4. Upon getting the response, Oauth server will respond to filter. For a valid user it respond with a token.
5. If a token is returned from OauthServer (in case of a valid user).

* Filter will create an entry in the LoginMap ( where it keeps the token as key and UserPojo as an object).

For an invalid user ( when OauthServer responded with validation error message).

1. For a valid user Filter will respond to client with a token else Error message will be returned.

OauthServer: In the above framework, Oauth Server works as an independent module, which generates an unique token with a token expiration period.

Filter: Filter intercepts each of the client request before it hits to cloud server, so that validation can be accomplished in Filter.

LoginMap : It’s a map which contains token as key and UserPojo as value pair. When the user trying to access any service with a valid token , cloud server be identified the user referring the LoginMap.

**[Post login scenario]**

Web Servcie / Cloud Server

Oauth Server

FILTER

CLIENT/MOBLE APP

Client accessing a service URL with a token

Filter access the Oauth server to confirm on token expiration.

Oatuh server respond to filter with appropriate message. For an expired token filter removes the token from LoginMap and ask the user to login again.

For a valid token filter allows the request to access cloud server

Here the response from the cloud received by the filter.

If client accessing with an invalid or expired token filter ask the client to login, else hands over the client with cloud response.

Post login scenario.

1. User accessing any cloud service url with a token.
2. Filter will intercept the user request, fetch the token and talk to Oauth Server for authenticating the token.

* If the token is authenticated , filter will allow access to cloud
* For an invalid token it will respond with a validation message
* For an expired token it will, remove the entry from the LoginMap and ask the user to login again.

1. Filter receives the response from the cloud server and for an expired token it removes the entry from LoingMap.
2. For a valid token, Filter allows the request to access the cloud server.
3. Filter receives the client response.
4. Filter respond to client with cloud response.