International Journal of Computer Science and Engineering (IJCSE)
ISSN(P): 2278-9960; ISSN(E): 2278-9979
Vol. 3, Issue 6, Nov 2014, 1-10

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

UNDERSTANDING THE WORKING OF OAUTH PROTOCOL

MRITUNJAY OJHA[1], SAYALI GOREGAONKAR[2] & POOJA DURGE[3]

[1]Assistant Professor, Department of Computer Science, Fr. C. R. I. T, Vashi, Navi Mumbai, Maharashtra, India

[2,3]Student, Department of Computer Science, Fr. C. R. I. T, Vashi, Navi Mumbai, Maharashtra, India

ABSTRACT

O Auth stands for Open Authorization. It's a free and open protocol, built on IETF standards and licenses from the Open Web Foundation, and is the right solution for securing open platforms. OAuth 2.0 framework was published in October of 2012, is an open standard used for providing authorization to third-party applications. The OAuth is implemented with help of authorization token with limited rights, which the user can revoke at any time should they become suspicious or dissatisfied with the app they're using to access your application. This method allows an application provider to gain access to selective APIs without the need for entire user-data. OAuth 2.0 was succeeded by its 1.0 standard which soon got replaced because OAuth 2.0 provided better non-browser based application support, less complicated signatures (no special parsing, sorting or encoding) i.e. SSL is required for all communication, only one security token on every API call and more security features. OAuth 1.a overcame with all the problems associated with 1.0 and is more secure than 2.0. OAuth is a widely used standard by several Resource Providers and social networking sites. This paper deals with the working of OAuth Protocol.

KEYWORDS: Authorization Server, Access Token, Consumer (Client), Request Token, Resource Owner (User), Refresh Token, Resource Server (Service Provider)

INTRODUCTION

As the technology advances the users prefer to have personalized browsing experience and so a large number of websites use social networks (for social sign on, social sharing social integration of APIs etc.). Using these social networking web applications, the websites can keep a track of the virtual world of the users i.e. the social data, images shared with the help of APIs used to establish connection. Access to these APIs is mediated by an authorization protocol that ensures that only websites that a user has explicitly authorized may access her social data. Web authorization protocol like OAuth is used by many internet giants like Facebook, Google, Microsoft, LinkedIn, and Amazon among many others for API access to social APIs. Thus it is no longer uncommon to see websites supporting a variety of login options using different social net-works.

OAuth is an open standard for authorization. OAuth provides a method for clients to access server resources on behalf of a resource owner (such as a different client or an end-user). It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections.

OAuth allows notifying a resource provider that the resource owner (e.g. the user) grants permission to a third-party access to their information. This can be explained as, consider you have an existing GMail account. You decide to join LinkedIn. Adding all of your many, many friends manually is tiresome and error-prone. You might get fed up

half-way or insert typos in their e-mail address for invitation. So you might be tempted not to create an account after all.

Facing this situation, LinkedIn has the Good Idea(TM) to write a program that adds your list of friends automatically because computers are far more efficient and effective at tiresome and error prone tasks. Since joining the network is now so easy, there is no way you would refuse such an offer, now would you? Without an API for exchanging this list of contacts, you would have to give LinkedIn the username and password to your GMail account, thereby giving them too much power.

This is where OAuth comes in. If your GMail supports the OAuth protocol, then LinkedIn can ask you to authorize them to access your GMail list of contacts. OAuth allows for:

- Maintain same space between numbering and text. Different access levels: read-only VS read-write. This allows you to grant access to your user list or a bi-directional access to automatically synchronize your new LinkedIn friends to your GMail contacts.
- Access granularity: you can decide to grant access to only your contact information (username, e-mail, date of birth, etc.) or to your entire list of friends, calendar and what not.

It allows you manage access from the resource provider's application. If the third-party application does not provide mechanism for cancelling access, you would be stuck with them having access to your information. With OAuth, there is provision for revoking access at any time.

Nowadays, more and more web developers are using open standards like RESTful APIs, OpenID for authentication as well as OAuth for authorization. OAuth gained a lot of interest and has become the IETF (Internet Engineering Task Force) standard after the launch of its 2.0 standard. It is currently the most widely supported protocol for API authorization, especially for REST, AJAX, and JSON-based web-sites.

OpenID

OpenID is an open standard. Certain web-sites use OpenID to authenticate its users using third party service.OpenID provides an identity assertion which allows its users to consolidate their digital identities, thus eliminating the need for webmasters to provide their own ad hoc services.It also facilitates the transfer of user attributes, such as name and gender, from the OpenID identity provider to the relying party (each relying party may request a different set of attributes, depending on its requirements).

The user created accounts with their Open ID credentials are used by them to sign-in onto any web-site providing Open ID authentication. Eg. You have created a URL for a service/website i.e. if you have a personal blog at wordpress.com you might use your yourname.wordpress.com URL. When you go to a new web site that accepts Open ID logins, just click on the Open ID button, type in your preferred URL and click to login. The Open ID protocol does not rely on central authority to authenticate user’s identity.

Table 1: Comparison between OpenID and Oauth

Parameters	Open Id	Oauth 1.0
Created for	To provide federated authentication i.e. letting third party authenticate users for you.	To authorize the users without them sharing their username and password with the third party applications.
HTTP Basic Credentials (i.e. username and password).	Cannot be used to provide API.	Can be used to provide API
Authorization access is provided by the user	To their digital identities.	To their protected resources.

OAUTH 1.0

The OAuth protocol was designed as a three-legged authorization that involves an end-user, a consumer service requesting authorization, and a server hosting the end-user's resources.

Specification Structure

The OAuth 1.0 specification consists of two parts [6].

The first part defines a redirection-based browser process for end-users to authorize client access to their resources. This is done by having the users authenticate directly with the server, instructing the server to provision tokens to the client for use with the authentication method.

The second part defines a method for making authenticated HTTP requests using two sets of credentials, one identifying the client making the request, and the other identifying the resource owner on whose behalf the request is being made.

The following is an outline of the OAuth 1.0 protocol specification:

- **Introduction**

It provides a quick overview of the specification and its objectives. The terminology sub-section defines the terms used and their relation to the HTTP specification (this guide provides a more detailed description). The example sub-section provides a full walk-through, describing a typical use case of a user sharing photos on one site and looking to print them on another.

- **Redirection-Based Authorization**

The authorization flow is what most people think of when they talk about OAuth. It is the process in which the user is redirected to the server to provide access. This section describes the three steps used to request and grant access: Obtaining Temporary Credentials, Requesting Resource Owner Authorization, and Obtaining Token Credentials.

- **Authenticated Requests**

In order for the client to obtain a set of token credentials and use it to access protected resources, the client must make authenticated HTTP requests. This section describes how the client makes such requests, how the server verifies them, and the various steps and cryptographic options available. Most of this section is dedicated to the construction of the signature base string, the normalized version of the request used for signing.

- **Security Considerations**

This often-overlooked section is a must-read for any developer writing client or server code. It provides a comprehensive (but never complete) list of issues which can greatly impact the security properties of any given implementation. Developers should read this list before starting any OAuth related project, and read it at least once more when they are done to review.

MIGRATING FROM OAUTH 1.0 TO OAUTH 2.0

This section describes how to migrate your application from OAuth 1.0 to OAuth 2.0, without the need for end-user intervention. After migration, your application can stop using the OAuth 1.0 library and can access resources using OAuth 2.0 tokens.

Prerequisites for This Migration

The application must have a valid OAuth 1.0 access token (i.e. application has a valid grant).

Also application must have an OAuth 2.0 check justification for this paragraph client ID and client secret. They can obtain these OAuth 2.0 credentials by registering the application as a web app. You present both the OAuth 1.0 and OAuth 2.0 credentials in a single request; there is no other binding of the two grants. The idea is to exchange an OAuth 1.0 grant for an OAuth 2.0 refresh token.

The basic flow is as follows:

- The process is triggered when an OAuth 1.0 consumer (your application) sends a migration request to server's OAuth 2.0 token endpoint.
- The credentials in the migration request are validated.
- An OAuth 2.0 refresh token is issued to your application in the response to your migration request.
- Your application uses OAuth 2.0 flows to exchange the refresh token for an OAuth 2.0 access token. (Note that when the refresh token is used, the OAuth 1.0 access token will expire in 1 hour.)

As soon as your application obtains the OAuth 2.0 access token, it can drop the OAuth 1.0 token and use OAuth 2.0 tokens to access users' resources.

OAUTH 2.0

The next evolution of the OAuth protocol is OAuth 2. 0, it is not backwards compatible with OAuth 1.0. OAuth 2.0 mainly focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices

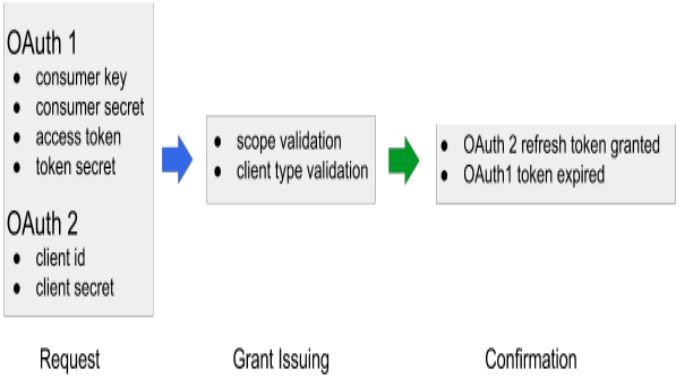

Figure 1: OAuth Process Flow [6]

OAuth 2.0: Browser-Based API Authorization

The goal of the OAuth 2.0 protocol is to enable third party clients to obtain limited access, on behalf of a resource owner, to the API of a resource server.

The protocol involves five parties.

- Authorization Server
- User Agent
- Resource server : It allows access to its resources over the web on receiving an access token issued by a trusted authorization server
- Resource owner: He owns data on the resource server, has login credentials at the authorization server, and uses a user-agent (browser) to access the web.
- **Client Website:** It needs to access data at the resource server, and whose application credentials are registered at the authorization server

For example Facebook uses both the authorization server and resource server. OAuth 1.0 was designed to unify previous authorization mechanisms implemented by Twitter, Flickr, and Google. However, it was criticized as being website-centric, inflexible, and too complex. In particular, the cryptographic mechanisms used to protect authorization requests and responses were deemed too difficult for website developers to implement.

OAuth 2.0 is designed to address these shortcomings. The protocol specification defines several different flows or protocol configurations, two of which directly apply to website applications.

The protocol itself requires no cryptographic mechanisms whatsoever and instead relies on transport layer security (HTTPS). Hence, it claims to be lightweight and flexible, and has fast emerged as the API authorization protocol of choice, supported by Microsoft, Google and Facebook, among others.

We next describe the two website flows of OAuth 2.0, their security goals, and their typical implementations.

OAUTH 1.0a

OAuth 1.0 was vulnerable to session fixation that is if you make a user approve ‘oauth_token’ issued for your

account, then you can use the same 'oauth_token' to sign-in his account on client website. OAuth 2.0 is compatible with rest of the social networking sites as it is less complex and works with the help of access_token, client_id of this token and current user_id; which makes the authorization flow insecured. These vulnerabilities were fixed in OAuth 1.0a.

It uses access_token for event handling. When this token expires or user revokes the permission to access the application, the application must go through the authentication flow again in order to obtain a new access_token.

TOKENS

Clients can gain access to his resources from the resource server with the help of access token. The access token replaces the authorization credentials (username, password) with the single token which is known only by the user and the server.

OAuth Authentication is done in three steps.

- The Consumer obtains an unauthorized Request Token.
- The User authorizes the Request Token.
- The Consumer exchanges the Request Token for an Access Token.

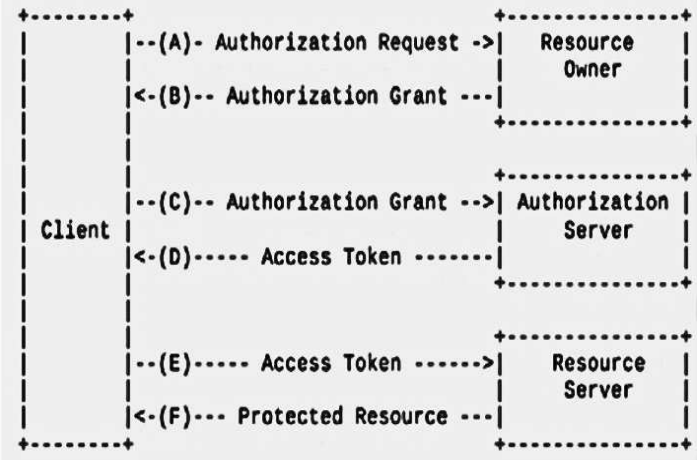

Figure 2: Flow of Access Token [9]

- Client requests for authorization to Resource owner
- Resource owner grants the authorization to the client
- Client asks permission to gain access to protected resources from the authorization server.
- The authorization server gives client access token, which is in turn used as a key to gain access from resource provider.
- Client asks for gaining access to the resources from the resource server by providing the access token

The resource server validates this token and on validation provides the client with the protected resources.

OAUTH CREDENTIALS

OAuth uses three type of credentials:

- Client credentials (consumer key and secret) It is used to authenticate the client. It allows the server to collect client information using its services, special treatment is provided for some clients such as throttling-free access or provide the resource owner with more client information.
- Temporary credentials (request token and secret)

  They are used to identify the authorization request.
- Token credentials (access token)

They are used instead of resource owner's credentials. It authorizes the server to issue a special class of credentials to the client which represents the access grant given to the client by resource owner. They are usually limited in scope and duration.

BLOCK DIAGRAM

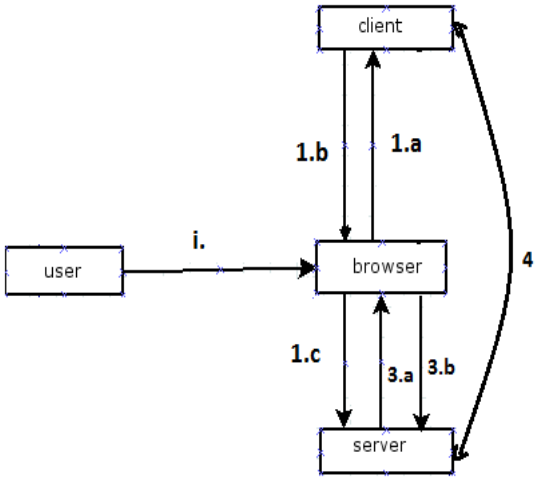

Figure 3: Front-End of the Proposed System

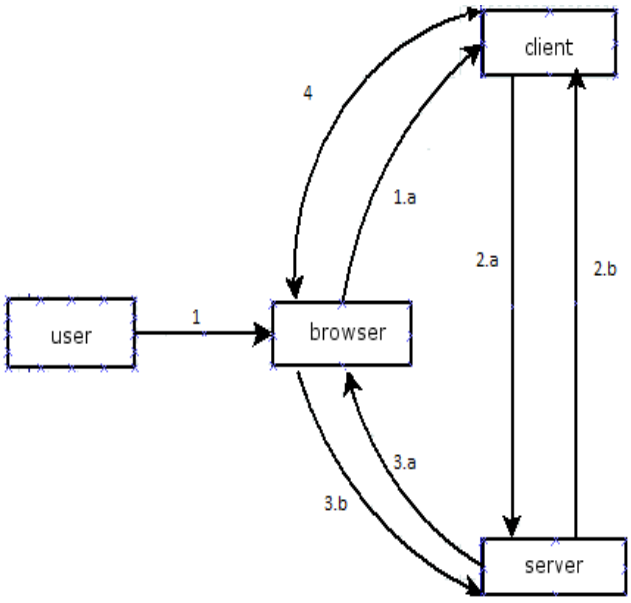

Figure 4: Back-End of the Proposed System

References for above block diagrams.

i-user accesses client website via web/internet browser

1. a-Browser hosts client website

1. b-Client use web browser for redirection

1. c-browser redirects to server website

3. a-server website sends a acknowledgement/ approval message to user via browser

3. b-user responds to the message received

AUTHORIZATION CODE FLOW

The authorization code flow, also known as Explicit grant flow or Web Server flow, is used by client websites wishing to implement a deeper social integration with the resource server by using server-side API calls. The client should have a shared secret with the authentication server, also the access token should be retrieve on the server site by the client.

In the diagram below text is not visible properly

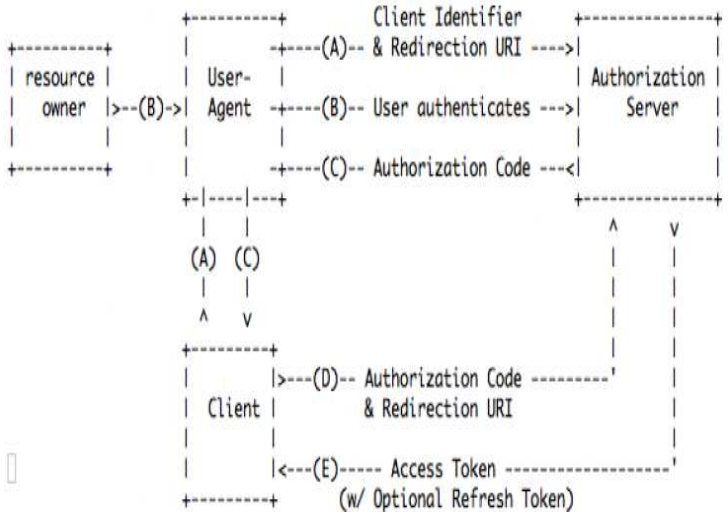

Figure 5: Authorization Code Flow[9]

The flow illustrated in above figure is as follows

- The client initiates the flow by redirecting the resource owner user/agent to the authorization endpoint.
- The authorization server authenticates the resource owner and grants or denies client's access request.
- Depending on the resource owner's access the authorization server redirects the user agent back to the client.
- The client requests an access_token from the authorization server token endpoint by including the authorization code received in previous step.
- The authorization server authenticates the client, validates the authorization code and ensures that the redirection URI received matches to URI used to redirect the client in step C. If valid, the authorization server responds that with an access token and, optionally a refresh token.

PROPOSED SYSTEM

The system proposed is to demonstrate the implementation of OAuth protocol. As OAuth is a three legged protocol, it requires a client side, a server side and a user. The server side would have web application known as 'Whatzzdat'. The client side would be another web application 'Rhythm House' and the user would be us.

The server side web application would be a website which would give information about the latest movies released, songs included in them, songs belonging to a different bands, popular songs. The audio/video clip of the respective songs can also be played. The user can search for the required movies and songs. The user can register with our site if he wants to post reviews. The user who is not registered with us would not be able to post reviews, but he can see the page and listen and watch the video clips. In order to download these clips the user would be redirected to our client website.

The client side website user can search for the song clips and even listen to them. If the user wants to download the song clip he can do so by selecting the download option. He would be asked to either register with our site or log in with server website Whatzzdat.

The user if already logged on with the server website he would be redirected to the client website and he can download the song.

When the user who is not logged in, wants to download he would be initially redirected to the client site. Therefore, before the downloading starts he would be asked to log in using server or client account. When the user selects the server log in, the client uses its OAuth credentials to begin the authentication process for the user. The client redirects user temporarily to server to log in. (client never sees user's server log in credentials). The page displayed by the server asks if the application i.e. the client can access server on the user's behalf. Note that the server shows the user what rights the client website is asking for, and importantly what the client website will not be able to do i.e. for our project the client will not be able to access the user's direct comments, his profile, his logs and his server log in credentials.

If the sign in is successful, the client uses its own OAuth credentials (token) to retrieve credentials for the user (i.e. the valet key that allows the client to access the server on user's behalf). The client stores user's credentials along with user's account. They allow him to use client website.

CONCLUSIONS

As the web grows, more and more sites rely on distributed services and in today's integrated web, users demand more functionality in terms of usability, cross- platform integration, cross-channel experiences and so on. It is up to the implementers and security professionals to safeguard user and organizational data and ensure business continuity, which can be done by implementing the OAuth protocol. The implementers should not rely on the OAuth protocol alone to provide all security measures, but instead they should be careful and consider all avenues of attack exposed by the protocol, and design their applications accordingly.

REFERENCES

1. "OAuth.pdf", Chetan Bansal, Karthikeyan Bhargavan,Antoine Delignat-Lavaud, Sergio Maeisy, Project-Team PROSECCO ,Research Report n° 8287 , April 2013
2. " OAuth - The Big Picture.pdf", Greg Brail & Sam Ramji,Apigee
3. http://hueniverse.com/OAuth/guide/workflow/
4. http://hueniverse.com/2010/05/introducing-OAuth-2-0/
5. https://developers.google.com/accounts/docs/OAuth
6. http://stackoverflow.com/questions/4201431/what-exactly-is-OAuthopen-authorization
7. http://en.wikipedia.org/wiki/OpenID
8. http://tools.ietf.org/html/rfc5849
9. http://OAuth.net/code/
10. http://OAuth.net/code/1.0a
11. http://stackoverflow.com/questions/4201431/what-exactly-is-OAuthopen-authorization
12. http://hueniverse.com/2009/04/explaining-the-OAuth-session-fixation-attack
13. homakov.blogspot.com.es/2013/03/oauth1-oauth2-oauth.html?m=1
14. aaronparecki.com/articles/2012/07/29/1/oauth2-simplified
15. https://developer.linkedin.com/oauth-10a-overview