Hi everyone, I’m Venugopal, I’ll be presenting on oAuth today. I’ll go through the concept followed by a brief demo of how we could access a reddit user’s data using oAuth.

oAuth is used in a variety of settings but before I get to that the name suggests it has something to do with Auth, what does the Auth stand for? It stands for Authorization and not Authentication.

It was meant not for a service to authorize a person, but a service to authorize another service as I’ll explain this now.

SLIDE 2

Valet key example

The closest parallel to oAuth that one could think of is the classic valet key example.

Let’s suppose you have a very fancy and powerful car and go out to a place with valet parking.

You don’t want the valet speeding off with the car or going through your personal effects in the glove box etc.

Thus, these cars usually have a valet key or a valet mode which restricts the power and functions of the car, thereby granting only limited access to the valet.

That’s what oAuth does! it is a framework to gain limited access to a user’s account.

SLIDE 3

Let’s see how this would work with another example:

Let’s assume you run a social media website service with users, you need the user’s friends and posts from Instagram, how would you do it?

One way could be you ask the user to give their Instagram credentials and you promise you won’t do anything else with their account. That’s not happening!

The other way is you ask the user to upload their posts manually. The problem with that is that your user may be lazy or may not be able to perform the task, for instance uploading a list of thousands of followers’ names and account handles is kind of unreasonable.

This is where oAuth comes in. In order for services to access each other on behalf of the user there was a standard created called oAuth. Initially there was oAuth 1.0 but the most widely used service today is oAuth 2.0. I’ll cover some of the key differences shortly.

SLIDE 4

oAuth Terminology

Let me cover some key terminologies that you would encounter when working with oAuth:

Term 1: The Resource or the Protected Resource: This is what we are seeking, for instance the Instagram photos in this case are the resource with limited access.

Term 2: The Resource Owner: This would be our user who owns the protected resource. The formal definition would be an entity who is capable of granting access to a protected resource.

Term 3: The Resource Server: This refers to the servers where the protected resource is stored.

Term 4: The Client: This is where it gets a little confusing, our social media website or the third-party service has the unfortunate name of being called the client.

An application making protected resource requests on behalf of the resource owner and with its authorization.

Term 5: The Authorization Server: Typically, the resource server has coupled with it an authorization server which makes sure that the service is authorized. The server that issues access tokens to the client.

SLIDE 5

oAuth Flow

There are several different types of flows in oAuth and each of those flows may be implemented in various ways. I’ll briefly go over three important ones here:

1. Authorization Code Flow: This is the flow we’ll see today, basically what happens here is that the client first makes a call to the resource authorization server requesting access to a resource. The authorization server then contacts the resource owner, i.e. the user and asks them whether to grant access to the requested resources, once our user signs in with their credentials and grants permission, first an authorization token is sent over to the client. The client then makes another call with this authorization token and that’s when an access token is granted to the client. The client can then make api calls to the resource server with this access token and get access to the resources.
2. Implicit Flow: This is a second kind of oAuth flow which is simpler but also less safe than the first flow. Here instead of sending an authorization token first, the authorization server directly sends over the access token. This sort of flow is used for JavaScript applications since any JS application that runs in the browser will save the access token in the browser which isn’t very safe to begin with. The access tokens generated this way are usually short lived too.
3. Client Credentials Flow: This is the kind of oAuth flow that’s used in a microservices like architecture or where the client is very trustworthy. Let’s say you have for your social media website various microservices, one for say ML tasks, one for handling information that’s publicly available, one to handle user data, etc. So microservice 1 might need a certain type of resource from another microservice 2 and so the access token it would be granted by microservice 2 would just restrict it only to be able to access that resource.

SLIDE 8

oAuth 1.0 versus oAuth 2.0

Some key differences between oAuth 1.0 and oAuth2.0 which was introduced in 2012 are:

**More OAuth Flows to allow better support for non-browser-based applications.** This is a main criticism against OAuth from client applications that were not browser based. For example, in OAuth 1.0, desktop applications or mobile phone applications had to direct the user to open their browser to the desired service, authenticate with the service, and copy the token from the service back to the application. The main criticism here is against the user experience. With OAuth 2.0, there are now new ways for an application to get authorization for a user.

**OAuth 2.0 no longer requires client applications to have cryptography.** There is no need now for cryptographic methods such as hash message authentication codes or HMAC hash tokens and request strings. With OAuth 2.0, the application can make a request using only the issued token over HTTPS. The get and post requests themselves are not secure but their secured by SSL or TLS

**OAuth 2.0 signatures are much less complicated.** No more special parsing, sorting, or encoding.

**OAuth 2.0 Access tokens are "short-lived".** Typically, OAuth 1.0 Access tokens could be stored for a year or more (Twitter never let them expire). OAuth 2.0 has the notion of refresh tokens. You'd use a refresh token to acquire a new access token rather than have the user re-authorize your application. The user can at any time revoke access making the refresh token defunct.

**Finally, OAuth 2.0 is meant to have a clean separation of roles between the server responsible for handling OAuth requests and the server handling user authorization.**

https://stackoverflow.com/questions/4113934/how-is-oauth-2-different-from-oauth-1

JWT

Since we have a secure oAuth token that cannot be tampered with and is trustable, one of the formats that may be used for such a token is a JSON Web Token since JWT is a format that can contain data within it but is also secure. It is verifiable and cannot be modified.