OAuth 2.0

Time before OAuth

Before OAuth was invented, a common way to give third-party application access to your account is to just provide them your **password**. This way of allowing third-party application accessing your account information of course have number of problems on the surface.

The application need to login as the "user" so they must store the password somewhere, and often just as plaintext, so if the application is somehow hacked then your account information can be easily stolen. In addition, the only way of revoking the access after you give the application your password is to change your password, which nobody likes to do.

Then many services realize how problematic this traditional way of user granting access to third-party application and people started developing ways to solve this problem.

OAuth 1.0 was formulated on August 2007 that was meant to solve this problem. Then OAuth 2.0 was started to simplify and clear up many of the problems that previous OAuth 1.0 had. Nowadays, OAuth 2.0 is the de facto standard for **authorization (giving third-party application the privilege to access private resources on behalf of a user)**.

How does OAuth 2.0 work

Keep in mind that OAuth 2.0 is an authorization protocol not an authentication protocol (verifies that the user is who he/she claim she is).

OAuth 2.0 uses Access Token. Access Token is a piece of data that represent the authorization (the go ahead) to access resources on behalf of the user.

OAuth 2.0 roles

Because everyone likes role-playing:

- `Resource Owner`: The user or the system that owns the protected resources and can grant access to them.
- `Client`: The application that requires access to the protected resources. In order to access the protected resources, the client must hold and provide the appropriate Access Token.
- `Authorization Server`: This is the server that receives request from the client that is asking for Access Tokens and issue them on successful authentication and consent by the resource owner. The authorization server give two endpoints, the authorization endpoint,

this handles the authentication and consent of the user, and the token endpoint, which is where the client will go to get grants (a set of steps that client has to do in order to get resource access authorization) there are different kind of grants to handle how Access Token are granted.
- `Resource Server`: The server that hold the protected user resources and will receive access requests from the client. It will check and validate the Access Token from the client before giving back the requested protected resources.

OAuth 2.0 scopes

Scopes defines the which resources that they are requesting, and whether it is read/write.

Flow of OAuth 2.0

1. First client request authorization from the authorization server, and providing the client id and client secret of the application itself
2. The authorization server authenticates the client and verifies that the requested scopes are permitted.
3. The resource owner interact with the authorization server to grant access (the agreement from the user saying that I allow this application to access my protected resources)
4. Then authorization server redirects back to the client with either authorization code or Access Token. With authorization code it will be used to exchange for an Access Token
5. With Access Token, the client can request access to the protected resource from resource server.

What is Bearer token?

When giving out Access Token to the client there are typically two choices. JWT or a Bearer token.

A Bearer token is just a string that is meant to be the Access Token.

On the other hand, a Java Web Token is a convenient way to encode and verify that you are indeed authorized.

Let's take a look at bearer token first, after you are granted a bearer token, that token is stored into a database on the server side. When the client makes the API call with that particular bearer token in order to access a user's protected resource, that token is submitted to the server and checked via a database lookup in order to verify that the client has been authorized to access those resources.

As you can see, Bearer token requires a database access every time the token is used.

On the other hand, Java Web Token contains a JSON payload that is signed and encrypted (the server has the public key to decrypt it to check it). The client can just submit the JWT and the server can just check it without a database lookup.

Revision #2
Created 2023-01-04 02:57:10 UTC by Tamarine
Updated 2023-01-06 21:08:14 UTC by Tamarine