Spring Boot -Spring Security -Oauth2-JWT

What is OAuth 2.0 and how it works?

**OAuth** doesn't share password data but instead uses authorization tokens to prove an identity between consumers and service providers. **OAuth** is an authentication protocol that allows you to approve one application interacting with another on your behalf without giving away your password.

What is OAuth 2.0 used for?

The **OAuth 2.0** specification defines a delegation protocol that is useful for conveying authorization decisions across a network of web-enabled applications and APIs. **OAuth** is **used in** a wide variety of applications, including providing mechanisms for user authentication.

What is OAuth flow?

**OAuth** 2 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean. ... **OAuth** 2 provides authorization **flows** for web and desktop applications, and mobile devices.

When should you use OAuth?

**You should** only **use OAuth** if **you** actually need it. If **you** are building a service where **you** need **to use** a user's private data that is stored on another system — **use OAuth**

What is OAuth in REST API?

**OAuth** is an authorization framework that enables an application or service to obtain limited access to a protected HTTP resource. To use **REST APIs** with **OAuth** in Oracle Integration, you need to register your Oracle Integration instance as a trusted application in Oracle Identity Cloud Service.

Is OAuth a SSO?

**OAuth** (Open Authorization) is an open standard for token-based authentication and authorization which is used to provide single sign-on (**SSO**). **OAuth** allows an end user's account information to be used by third-party services, such as Facebook, without exposing the user's password.

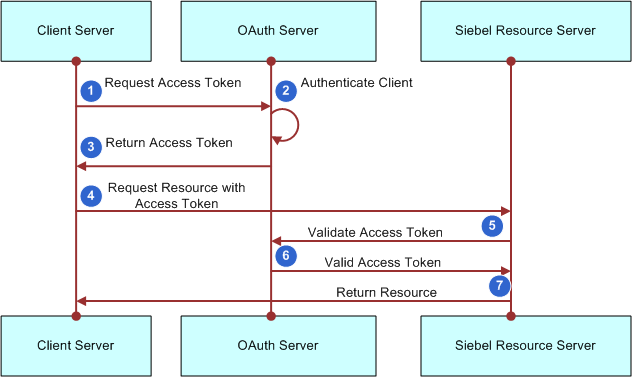
Is OAuth safe?

It's the most **secure** flow because you can authenticate the client to redeem the authorization grant, and tokens are never passed through a user-agent. There's not just Implicit and Authorization Code flows, there are additional flows you can do with **OAuth**. Again, **OAuth** is more of a framework.

What does OAuth stand for?

Open Authorization

OAuth, which stands for “**Open Authorization**,” allows third-party services to exchange your information without you having to give away your password.



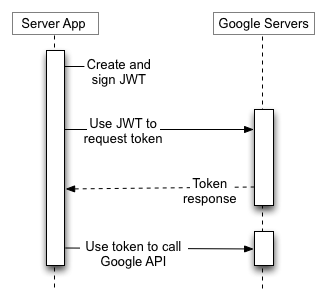
OAuth 2.0 defines a protocol, i.e. specifies how tokens are transferred, JWT defines a token format.

OAuth 2.0 and "JWT authentication" have similar appearance when it comes to the (2nd) stage where the Client presents the token to the Resource Server: the token is passed in a header.

But "JWT authentication" is not a standard and does not specify how the Client obtains the token in the first place (the 1st stage). That is where the perceived complexity of OAuth comes from: it also defines various ways in which the Client can obtain an access token from something that is called an Authorization Server.

So the real difference is that JWT is just a token format, OAuth 2.0 is a protocol (that may use a JWT as a token format).

Firstly, we have to differentiate JWT and OAuth. Basically, JWT is a token format. OAuth is an authorization protocol that can use JWT as a token. OAuth uses server-side and client-side storage. If you want to do real logout you must go with OAuth2. Authentication with JWT token can not logout actually. Because you don't have an Authentication Server that keeps track of tokens. If you want to provide an API to 3rd party clients, you must use OAuth2 also. OAuth2 is very flexible. JWT implementation is very easy and does not take long to implement. If your application needs this sort of flexibility, you should go with OAuth2. But if you don't need this use-case scenario, implementing OAuth2 is a waste of time.



Spring Boot Application with Spring Security Enabled using oauth2

1 understand what we need to create first we need to create and Authorization server and embedding Resource server inside authorization server.

2 So will be creating 2 service one will be authorization server along resource server second will be client server

Which will be accessing server

3 Authorization server will have token generated and Resource server will validate those tokens

4 only create 2 applications from source application we will query the authorization server get the token back and then query resource servers it will be be in same server but it will validate application with the tokens.

5 Create 2 applications 1 ->application with Authorziation server and resource server

1. >Client with which we access

Spring.io crete 2 apps

Pom.xml will have for oauth dependency

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-oauth2</artifactId>

</dependency>

First App: Spring\_Boot\_security\_oauth\_server

Dependency : web, Cloud Oauth2

Second App: Spring\_Boot\_Security\_client

Dependency : web, Cloud Oauth2, thymeleaf

\* Resource server requires REST endpoints so Spring-Boot-starter-web is mandatory dependency

I=> Need To add config For Resource Server => Spring\_Boot\_security\_oauth\_server

- create a config pacakage create class to config Resource Server => ResourceServerConfig

- @EnableResourceServer to make it has resource server

- @Configuration to make it load at Spring start up time

/\*\*

\*

\* **@author** SACHIN HS

\*

\* **@EnableResourceServer** to make this class Resource server

\* **@Configuration** to create bean and load automaticaly by spring boot

\*

\* need to override WebSecurityConfigurerAdapter to add security

\* to rest end points

\* will do authentcation here

\*/

@EnableResourceServer

@Configuration

public class ResourceServerConfig extends WebSecurityConfigurerAdapter {

@Autowired

private AuthenticationManager authenticationManager;

/\*\*

\* overide AuthenticationManagerBuilder

\*

\* autowire authenticationManager use inside AuthenticationManagerBuilder

\* using AuthenticationManagerBuilder we need say parentAuthenticationManager has authenticationManager

\* for now we do inmemeory not db fr temp

\*/

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.parentAuthenticationManager(authenticationManager).inMemoryAuthentication().withUser("sachin")

.password("password").roles("ADMIN");

}

/\*\*

\* override

\* http.requestMatchers().antMatchers("/login", "/oauth/authorize") : need to

\* match the request url match this and Authorize them for any request which

\* comes wch comes to this make them authenticated and show login page all every

\* to allow login page

\*/

@Override

protected void configure(HttpSecurity http) throws Exception {

http.requestMatchers().antMatchers("/login", "/oauth/authorize").and().authorizeRequests().anyRequest()

.authenticated().and().formLogin().permitAll();

}

}

II=> Need To add config For Authorization Server => Spring\_Boot\_security\_oauth\_server

- create class to config Authorization Server inside config pacakage => AuthorizationServerConfig

- @EnableAuthorizationServer to make it has Authorization server

- @Configuration to make it load at Spring start up time

/\*\*

\*

\* **@author** SACHIN HS

\* **@EnableAuthorizationServer** to make it has Authorization server

\* **@Configuration** to make it load at Spring start up time

\*

\* Need to Extend AuthorizationServerConfigurerAdapter so we can

\* overide Authorization part of Spring Provided int the way we

\* wanted

\*

\* - here we will override authorization server which spring

\* already have

\*

\* will do authorization here

\*/

@Configuration

@EnableAuthorizationServer

public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {

@Autowired

private AuthenticationManager authenticationManager;

/\*\*

\* Secure this Authorizaation server

\*

\* tokenKeyAccess who ever all can access token keys now permitting all :

\* security.tokenKeyAccess("permitAll()")

\*

\* need to validate token keys before token generated it need to be

\* authneticated :

\*

\* .checkTokenAccess("isAuthenticated()");

\*

\*/

@Override

public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {

security.tokenKeyAccess("permitAll()").checkTokenAccess("isAuthenticated()");

}

/\*\*

\*

\* For now use Inmemeory not DB

\*

\* SSO details of oauth need to be stored Authorzation server need to validate

\* data from some where so use inmemeory

\*

\* with client we need to provide clientId from which user will be accessing the

\* server : .withClient("clientId")

\*

\*

\* secret("secret") : will give secreat

\*

\* Need to provide Grant Type so will provide authorization\_code has grant

\* :authorizedGrantTypes("authorization\_code")

\*

\*

\* Scope of my authorization : .scopes("user\_info")

\*

\* AutoApprove to tokens generated : .autoApprove(true)

\*

\*/

@Override

public void configure(ClientDetailsServiceConfigurer clients) throws Exception {

clients.inMemory().withClient("clientId").secret("secret").authorizedGrantTypes("authorization\_code")

.scopes("user\_info").autoApprove(true);

}

/\*\*

\*

\* Overide AuthorizationServerEndpointsConfigurer with authenticationManager

\* provided

\*/

@Override

public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {

endpoints.authenticationManager(authenticationManager);

}

}

Now Both AuhorizationServer and Resource server are configured