Spring-security.xml

1. Authentication
2. Autherisation

Spring Security extends the Java standard security concept of an authenticated

principal (java.security.Principal),

Authorization typically involves two separate aspects that combine to describe the

accessibility of the secured system.

We can see that a component called the access decision manager is responsible for

determining whether a principal has the appropriate level of access, based on the

match between the authority possessed by the principal and the authority requested

by the resource.

To Enable Spring Security is a 3 step process

1. In web.xml specify following filter - DelegatingFilterProxy . It is a proxy class which delegates the control to a filter implementation which is defined as a bean named springSecurityFilterChain. This bean is an infrastructure internal bean to handle namespace configurations. Once this configuration is done, all the incoming requests enter the spring framework for security checks.

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filterclass>

org.springframework.web.filter.DelegatingFilterProxy

</filter-class>

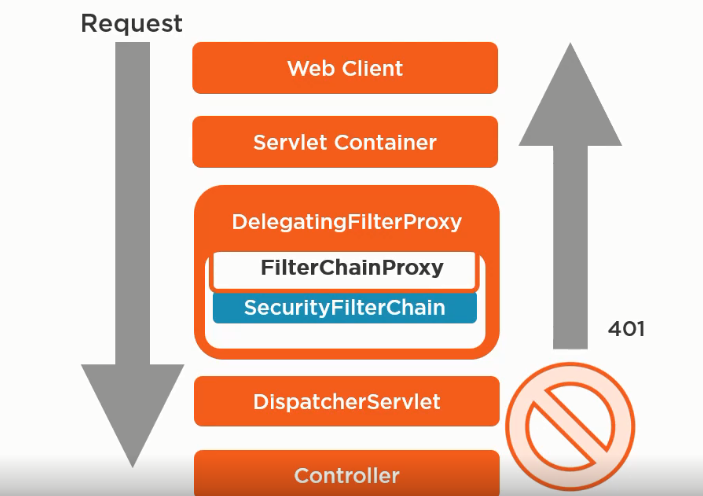
</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>



DelegatingFilterProxy ---------------🡪 calls DefaultSecurityFilterChain -------🡪 which invokes the following important list of filter

Creating filter chain: any request, [

org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@4f4c88f9,

[org.springframework.security.web.context.SecurityContextPersistenceFilter@3c91530d](mailto:org.springframework.security.web.context.SecurityContextPersistenceFilter@3c91530d),

[org.springframework.security.web.header.HeaderWriterFilter@149b0577](mailto:org.springframework.security.web.header.HeaderWriterFilter@149b0577),

[org.springframework.security.web.csrf.CsrfFilter@28f4f300](mailto:org.springframework.security.web.csrf.CsrfFilter@28f4f300),

[org.springframework.security.web.authentication.logout.LogoutFilter@7a4d582c](mailto:org.springframework.security.web.authentication.logout.LogoutFilter@7a4d582c),

[org.springframework.security.web.authentication.www.BasicAuthenticationFilter@7c52fc81](mailto:org.springframework.security.web.authentication.www.BasicAuthenticationFilter@7c52fc81),

[org.springframework.security.web.savedrequest.RequestCacheAwareFilter@732f6050](mailto:org.springframework.security.web.savedrequest.RequestCacheAwareFilter@732f6050),

org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@692dba54, [org.springframework.security.web.authentication.AnonymousAuthenticationFilter@6a756082](mailto:org.springframework.security.web.authentication.AnonymousAuthenticationFilter@6a756082),

[org.springframework.security.web.session.SessionManagementFilter@2ddb3ae8](mailto:org.springframework.security.web.session.SessionManagementFilter@2ddb3ae8),

[org.springframework.security.web.access.ExceptionTranslationFilter@436bd4df](mailto:org.springframework.security.web.access.ExceptionTranslationFilter@436bd4df),

[org.springframework.security.web.access.intercept.FilterSecurityInterceptor@3902bd2c](mailto:org.springframework.security.web.access.intercept.FilterSecurityInterceptor@3902bd2c)]

1. **SecurityContextPersistenceFilter** - so a SecurityContext can be set up in the SecurityContextHolder at the beginning of a web request, and any changes to the SecurityContext can be copied to the HttpSession when the web request ends (ready for use with the next web request)
2. **AbstractAuthenticationProcessingFilter**- UsernamePasswordAuthenticationFilter, CasAuthenticationFilter, BasicAuthenticationFilter etc - so that the SecurityContextHolder can be modified to contain a valid Authentication request token
3. **RememberMeAuthenticationFilter**, so that if no earlier authentication processing mechanism updated the SecurityContextHolder, and the request **presents a cookie that enables remember-me services to take place, a suitable remembered Authentication object will be put there**
4. **ExceptionTranslationFilter, to catch any Spring Security exceptions so that either an HTTP error response can be returned or an appropriate AuthenticationEntryPoint can be launched**
5. **FilterSecurityInterceptor, to protect web URIs and raise exceptions when access is denied**

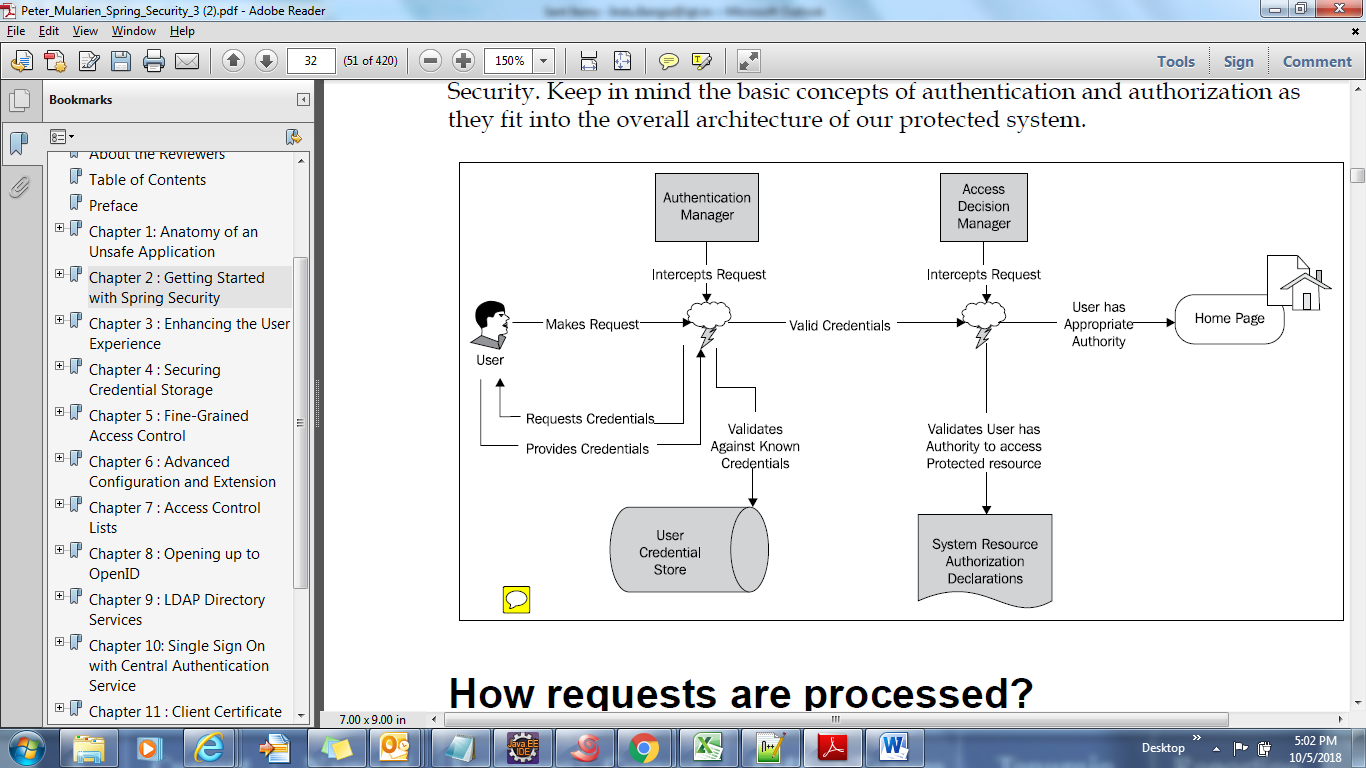
FormLogin -- UserNamePasswordAuthenticationFilter

httpBasic – BasicauthnticationFilter

DigestauthenticationFilter

JWT authentication Flow: - to authenticate RestAPI.

Oauth2 Flow-



User makes a request – Authentication Manager checks valid credential -🡪 access decision manager intercept request and check User has appropriate authority or not.

Oauth2 Login with Google and Facebook :-

Google uses OpenId Connect

Facebook uses Facebook Connect

Distributed Session Management

Side Car Proxy

Shared Library

API Gateway

Server Side and Single Page Application

Microservice Security Pattern

Advance Customization

Rest Services Security – One of the most popular authentication mechanism for rest services are Basic Authentication .

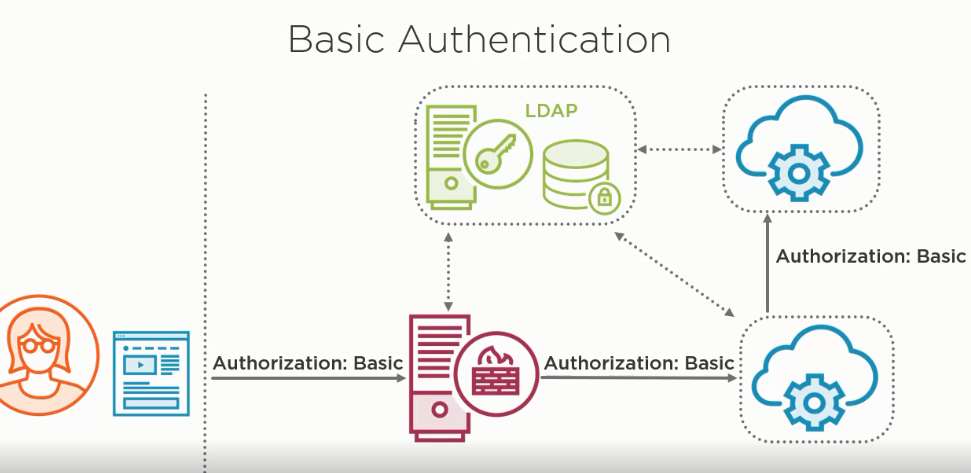
In Basic Authentication – Username:Password is encode in base64 and set in Header

Credential Transmitted in Header Like as follows

Header name :Authentication

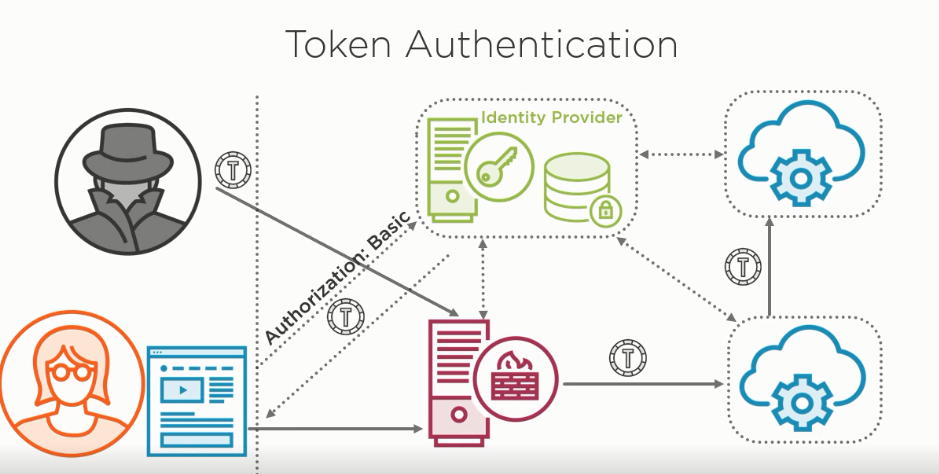
Header Value: Basic + Base64(username:password)

E.g Authorization: Basic Yui8786jkhdfkfio

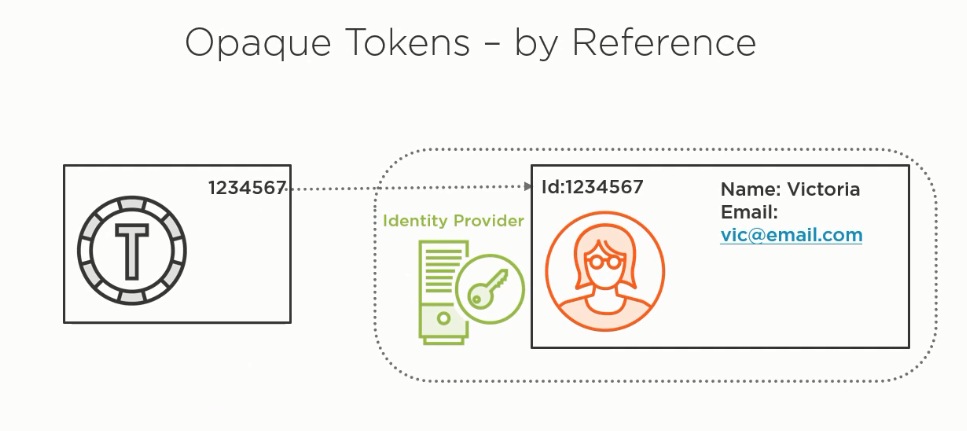


This approach has problem that it can not scale up. Because each service has to invoke Identity provider and each service has to send credential and too many services of handling credential and chances of leak are more.

To solve this user credential leak problem token approach has come. In this user sends the credential to Identity Provider and it sends back the token and then token is sends to each of the service. But scale up problem will not be solved because Token is opaque token each service has to sends the token to Identity provider to verify it by provider.

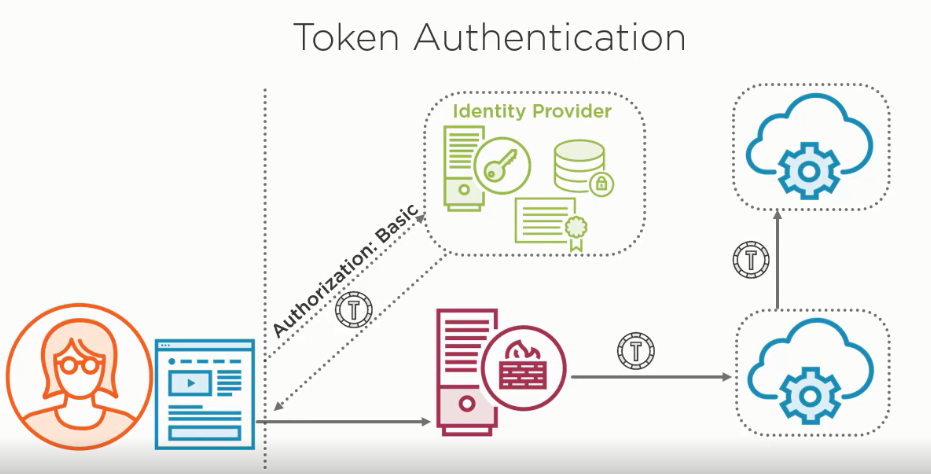


Opaque Token – by Reference Token - token contains reference id and identity provider contains complete detail by that reference id.

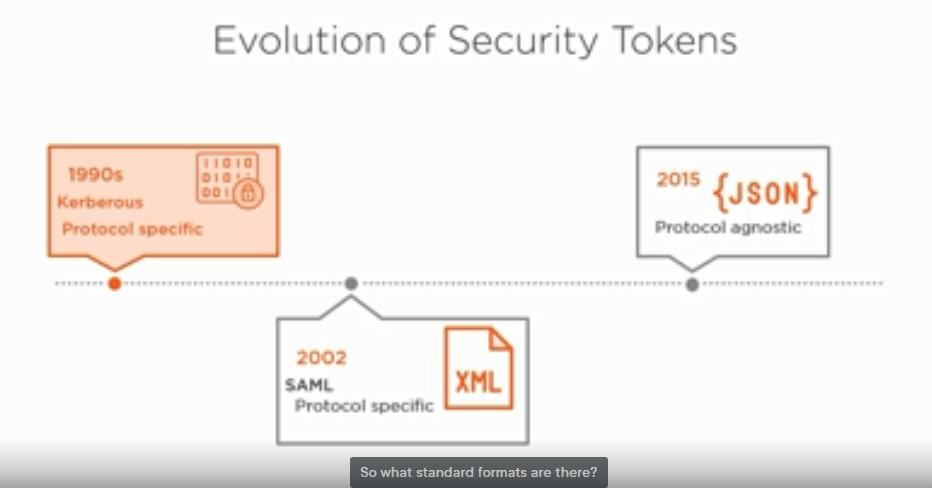


By Value Token - In this token all information like name, email, scope and expiry contains with in token and it eliminates the need to call Identitiy provider . These token are generally signed by Identity provider and each service uses public key of provider to ensure that it is valid





Standard Fomat of token



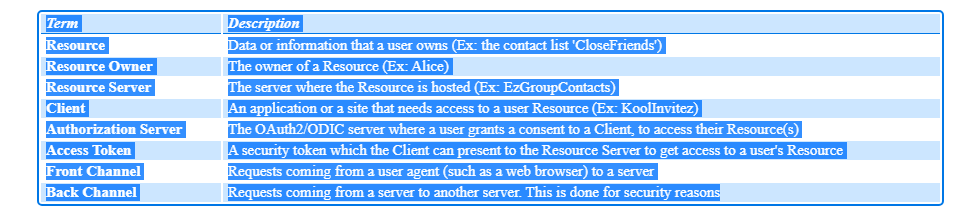
JSON Token Format



OAuth2 - OAuth2 is an open standard for users to grant access to their information on a site or application to another site, but without revealing their credentials.

OpenIdConnect - OIDC, on the other hand, is an extension on top of OAuth2, that is used to verify the identify of a user (authentication) in a standard way. As an example, there are many sites that do not have any user registration and rely on Google or Facebook for identity verification (authentication).

Some terms that will be useful in the context of OAuth2 and OIDC as follows:

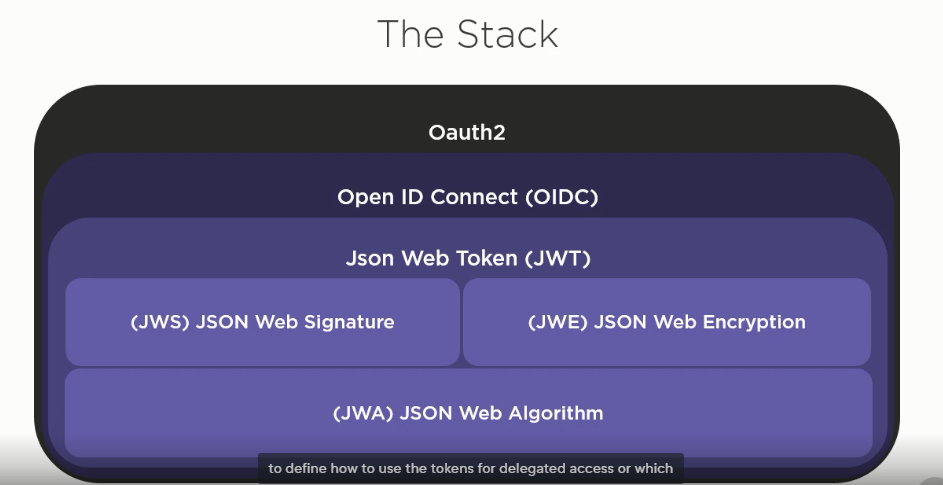


Resource --- Profile Info on Gmail,Facebook

Resource Owner – User ex Indu

ResourceServer – Gmail or Facebook and Okta

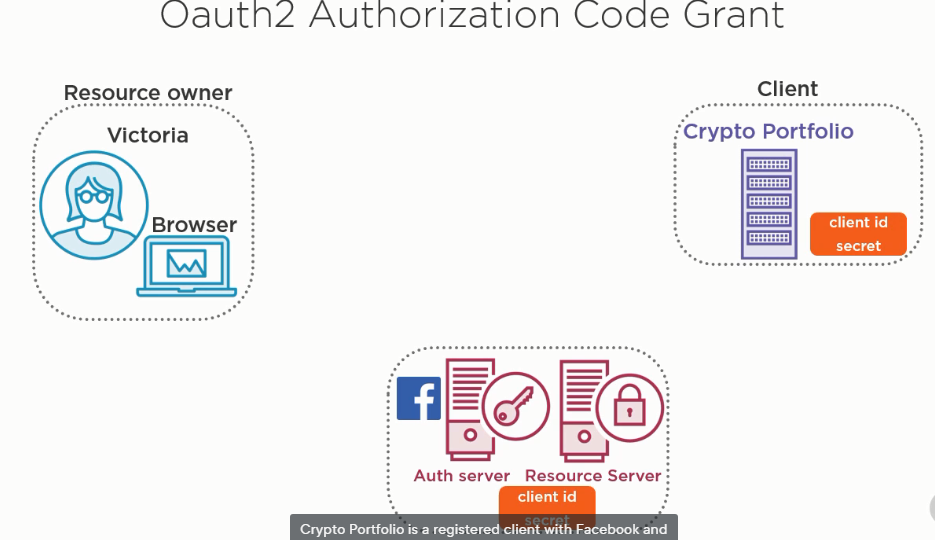
Client – Byjus and Vedantu.com (Which provides login with Gmail or Facebook)



Oathu2 – supports 4 type of Grant

1. Authorization Code Grant -example facebook login
2. OAuthClient Credential Flow - used be machine to machine authorization.
3. Implicit Grant Flow
4. OAuth2 Resource Owner Password Flow

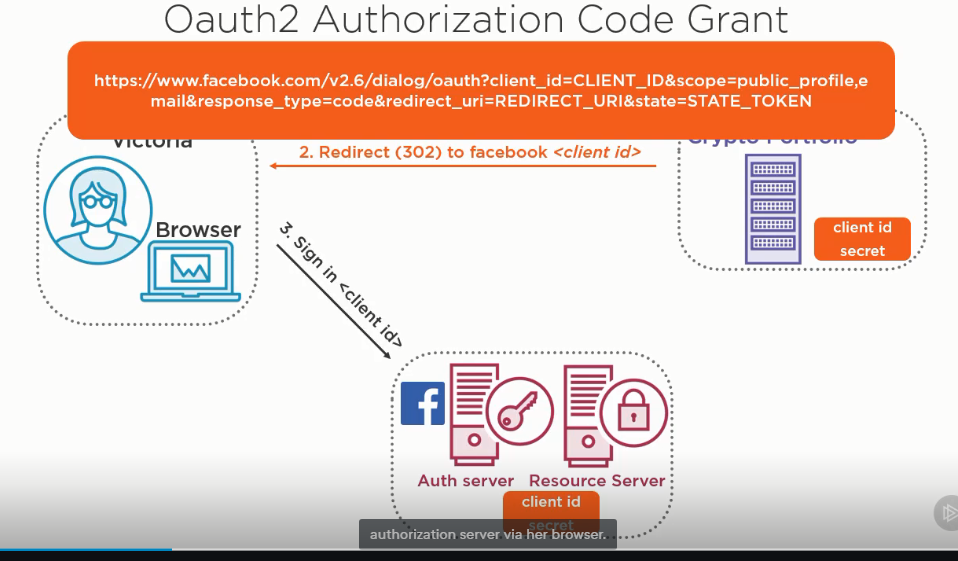
Authorization Code Grant :- Crypto Portfolio is registered with facebook. It has unique client id and secret. Crypo will have option “Login with Facebook”



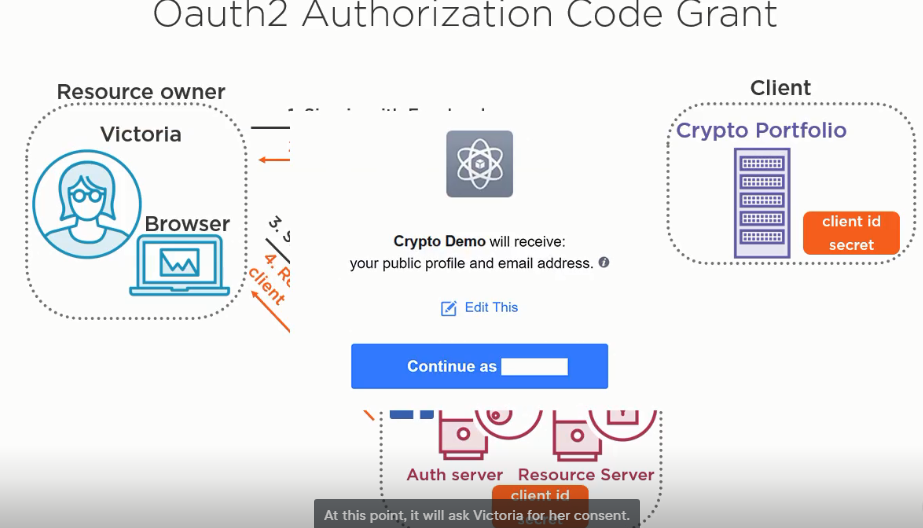
On click of “Login with Facebook” Crypto will redirect the user to facebook url with <client id> using following redirect url. Once facebook authenticate the user it display the consent form to user that whether Crypto can access the public\_profile and email

Scope=public\_profile,email

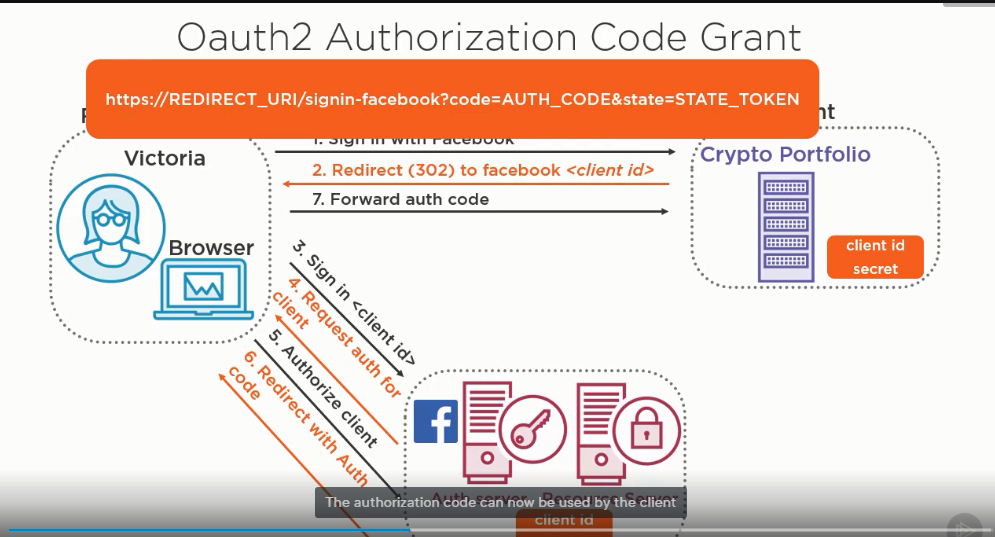
Step 1.



Step 2.

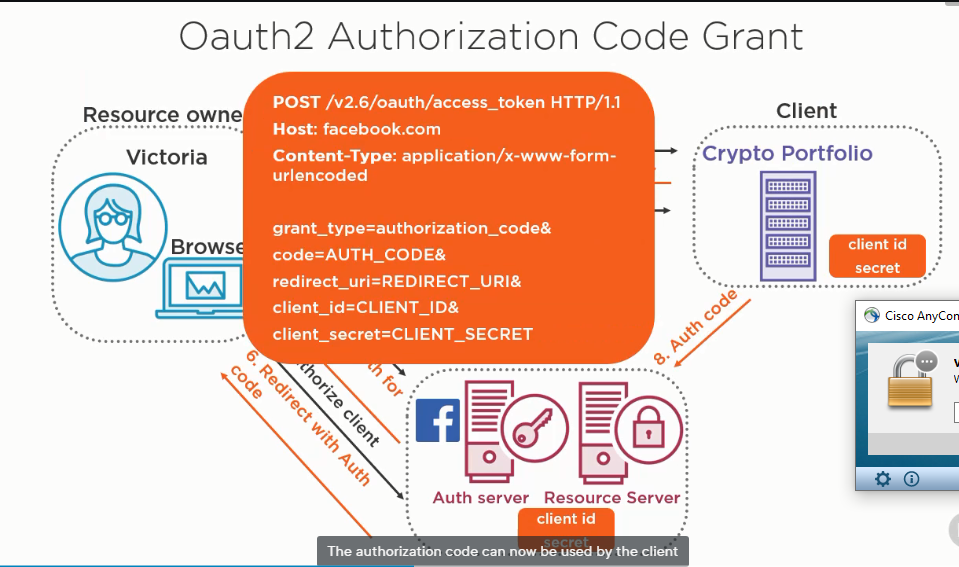
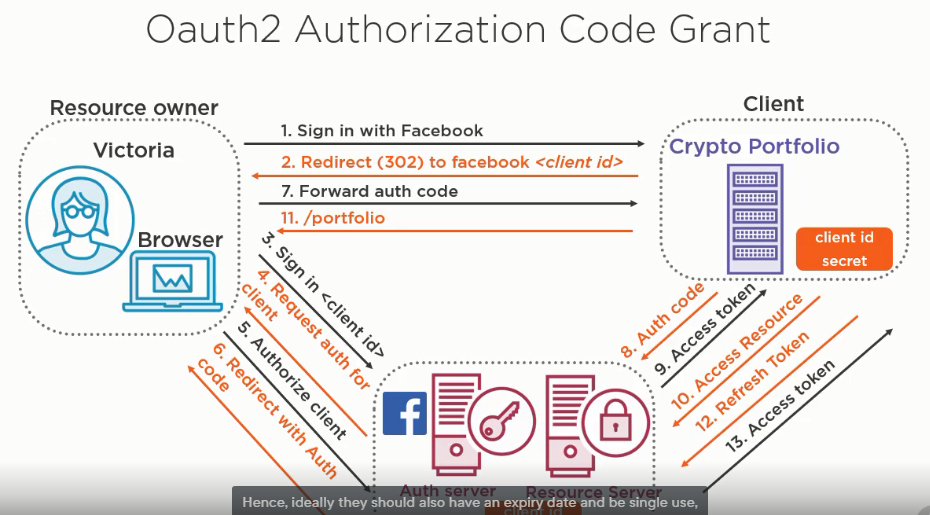


Once the User authorize the client ,Facebook returns the authorize code to client via browser with the URI specified by Crypto Portfolio



Authrization Code can now be used by the client to get the access token from facebook as define in below image.Access token has very short expiry time

The reason for using an authorization code and not passing the access token directly is that transmission via front channel between the authorization server,browser and client can not be completely relied upon as secure especially browser. It could be malicious plug-in.So, sending the bearer access token would be risky. On the other hand an authorization code without client secret is useless. And that exchange happens via back channel where the client directly goes to authorization server. There is one other token that can be returned along with access token that is the refresh token. Access token is bearer token and very less expiry time. This is to limit the damage of an access token being leaked. Often clients need the access token for longer and don’t want to keep bothering the user to reconsent each time the token expires. This is where refresh token comes in.Refresh token is used by client to get the new access token. Refresh token is securely stored on the client. If is leaked to malicious party to keep requesting the accesstoken. Hence, ideally they should also have expirydate and can be single use

The basic \***FUNDAMENTAL**\* flow of OAuth2 is referred to as the **Autorization Code** flow and all the other flows are variations of this basic flow. The Auorization Code flow works as follows:

* **1** :: the Resource Owner launches the Client to initiate the flow
* **2** :: the Client makes a request to the Authorization Server from the Front Channel for an authorization code (**/auth\_code**) passing in an URL to respond back to (**/callback**) at the Client
* **3** :: the Authorization Server redirects the Resource Owner to the Resource Server for user authentication and access grant
* **4** :: the Resource Owner enters valid credentials that is verified by the Resource Server and redirects back to the Authorization Server
* **5** :: the Authorization Server generates an Authorization Code and responds back to the Client with the authorization code on the Back Channel at the specified URL (**/callback**)
* **6** :: the Client makes a request to the Authorization Server for an Access Token through the Back Channel
* **7** :: the Authorization Server responds back to the Client with an Access Token on the Back Channel
* **8** :: the Client uses the Access Token to make a request to access the Resource at the Resource Server
* **9** :: the Resource Server responds with the data associated with the Resource