**OAUTH**

**The challenges with authentication and authorization:**

With earlier model of implementing security with authentication and authorization the UN/PW transmit over network leads to risk of theft.

UN/PW stored in DB and encrypted and validated in application also increase the vulnerability of any injection.

Scalability of the application will also open multiple end (when broken into micro services) point raising security risk.

When application broken down into micro services how to communicate with multiple components with same login/session and without transmitting UN/PW between multiple end points which also cause performance issue (This was not a problem in monolith application where there was only one endpoint and latency was low).

To address all these issues we need Tokens

**Token:**

Sharing credentials over network could make it vulnerable, anyone who breaks our network and also from persons inside our network like other employees and contractors.

So one way is encode the user details (Basic/BASE64) and pass it and each endpoint will authenticate the user details perhaps with some directory service/Identity provider LDAP.

But this method doesn’t scale well as each services needs to talk with identity provider to get the details and also reduce performance and increase risk of hack.

To solve this once a service will talk with IDP and get a Token (is a reference to a state in IDP) and pass this token and now services can communicate with each other with token and validate that token against the IDP.

Token sent along with http header just like Basic authentication but with “Bearer” prefix

But what if the token got hacked? It’s no different than a password and if you see also doesn’t solve the scaling problem as getting checked with IDP on each hub.

So its most important to token have an expiry time.

The above token are called **By Reference Token** where the state is present in IDP and referenced in each service to validate against IDP with token reference id.

Other type of token there called **By Value Token** which will have the state inside it (like Name, Email, Scope, Expiry time etc which can be decided by us) and will be signed by IDP and each service will validate with public key. This tokens solves the scaling problem if you see it doesn’t need to contact with IDP and state is present inside the token itself. But anyone can see the content if they have access to the token. Which can solve using encrypting the content.

So what we use is a **standard format** because we don’t want different components in our application communicating with each other in different format.



So the standard token are there:

* Kerberous (1990 protocol specific)
* SAML (2002 protocol specific)
* JWT (JSON Web Token 2015 protocol agnostic)

**JWT (JSON Web Token):**

JWT (collection of standard) is a base64 URL encoded, which could be easily sent in header, when we decode it looks below

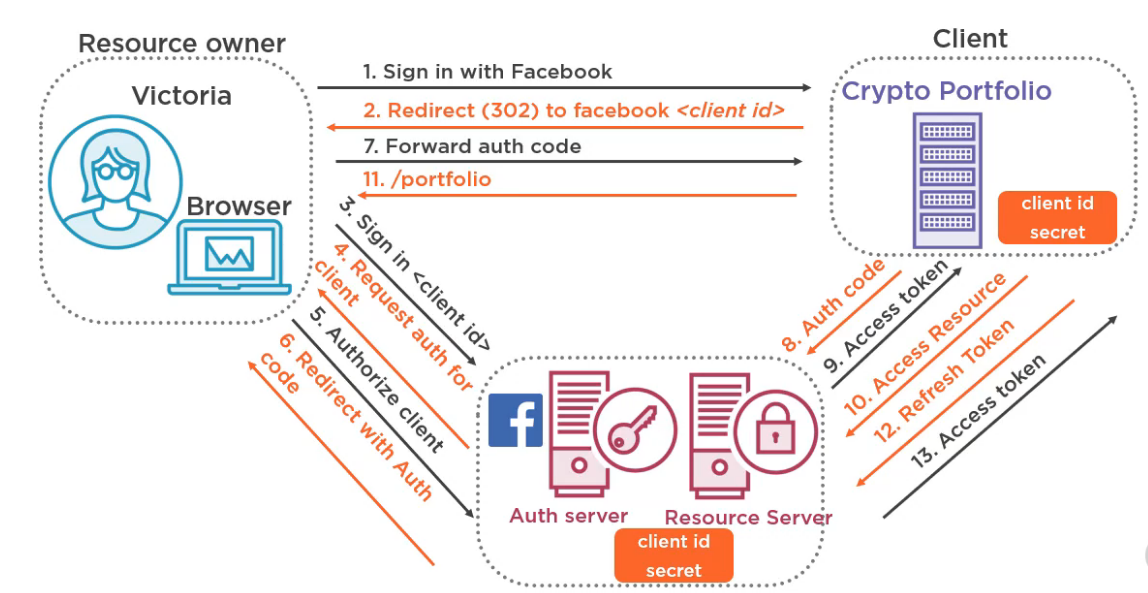


IDP creates a JWT sign it and sent it to client, JWT is a standard token format we need another standard to define how to use JWT, hence OAuth2

**OAuth2:**

OAuth2 is an open standard to protect resource called protected resource. There are multiple actors involved in the flow and a common and tested mostly secured flow is “Authorization Code Grant”.

**Authorization code grant:**



Client and Resource owner has a trust relationship with each other with client id secret.

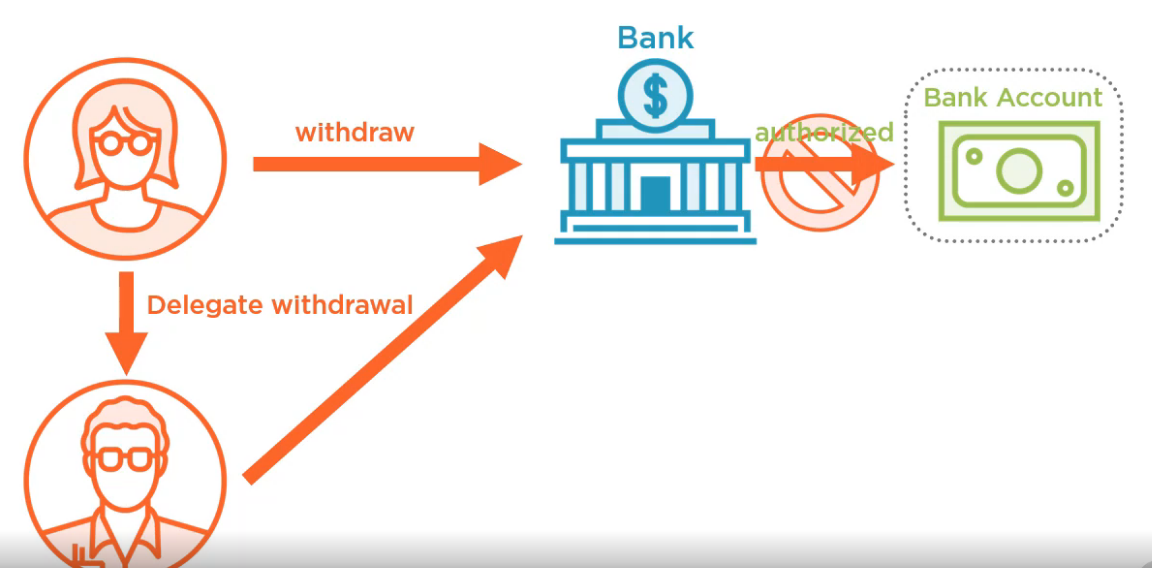
Resource server will send an Auth code to client which can be exchanged between Client and Resource server to exchange for “Access Token”.

Resource Server don’t sent access token to resource owner directly as the resource owner/browser considered to be most un secured.

Access Token has expiry time so there need to have Refresh token to re-grant access token. Refresh token also has expiry time to once expired a new Refresh token is issued.

**OAuth is not for Authentication:**

OAuth is for delegate authorization, what is delegate authorization



Here Victoria authorized Ron for withdrawal of amount but bank has a policy that the only user can do the transaction so even if bank completely verifies Victoria authorized Ron still bank won’t allow Ron to do the transaction as per policy called delegate authorization.

**Why we need OpenId connect:**

Because OAuth is not a federation which we can completely rely they will protect your identity

We already talked about Access Token which is By Value where user identity will be available if got leaked can be read by anyone.

Or developer at client side could read the access token and use the user name/email to identify the user for authentication.

The issue with above approach is Resource Server is not concerned if client is using the Access Token to do something else and if format of Access Token changed by Resource Server then Client logic will fail.

Access Token is purely for Resource Server not for Client then should not use it rather directly sent it to Resource Server for authorization.

If Token is stolen by hacker he/she can impersonate Victoria to access a different application.



To solve this many vendors provided implementation on top of oauth directly, so what we are missing a standard and there where OpenId comes to provide a single standard to provide authentication and identity to OAuth2.

**What makes ODIC great for Authorization:**

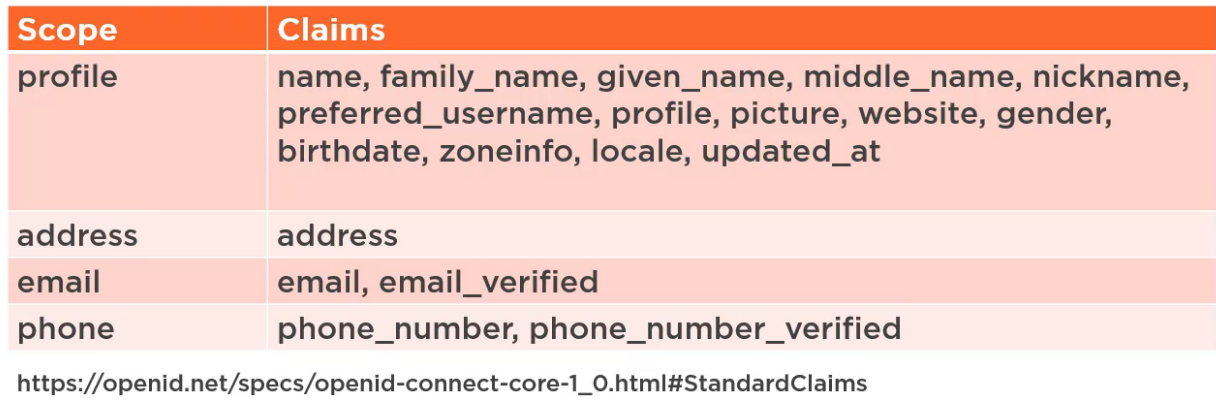
One of the feature of the ODIC connect is the identity token, which is a JWT got created with multiple mandatory and optional Authentication claims:

* Sub: Identity of the user which is unique and never re assignable
* Iss: Issuer, details of who is the issuing authority allow the client to validate whit whom it got validated
* Aud: Audience, Client for which it authenticate this, typically an IDP like Google supports thousands of client, our client can use this claim to confirm user was authenticated for our application.
* Exp: Expiry time (a standard format too)
* Iat: Issued at

ODIC also included some standard scope:

* Openid
* Profile
* Email
* Offline access
* Custom scope also supported

Key feature is for each and every scope there is default standard claim associated with it



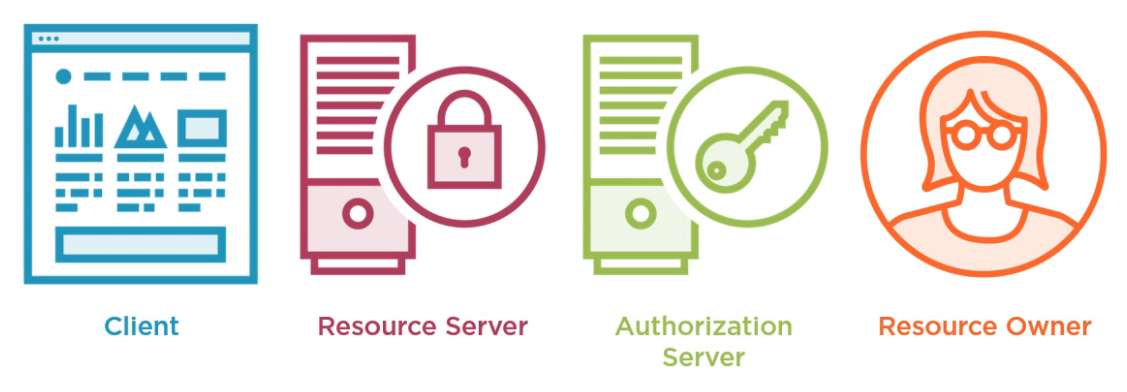
One thing the **endpoints** is same across with all ODIC provider, some of the key end-points

* GET/userinfo (to get all the user claims)
* Web Finder endpoint
* Provider metadata (Everything client needs to dynamically configure itself with provider to validate token)
* Client registration
* Session management (Check if user have a still an active session with authentication server, as the client also wants to logout the user from their end if it logged out from authentication server)

**OIDC Authorization code grant:**

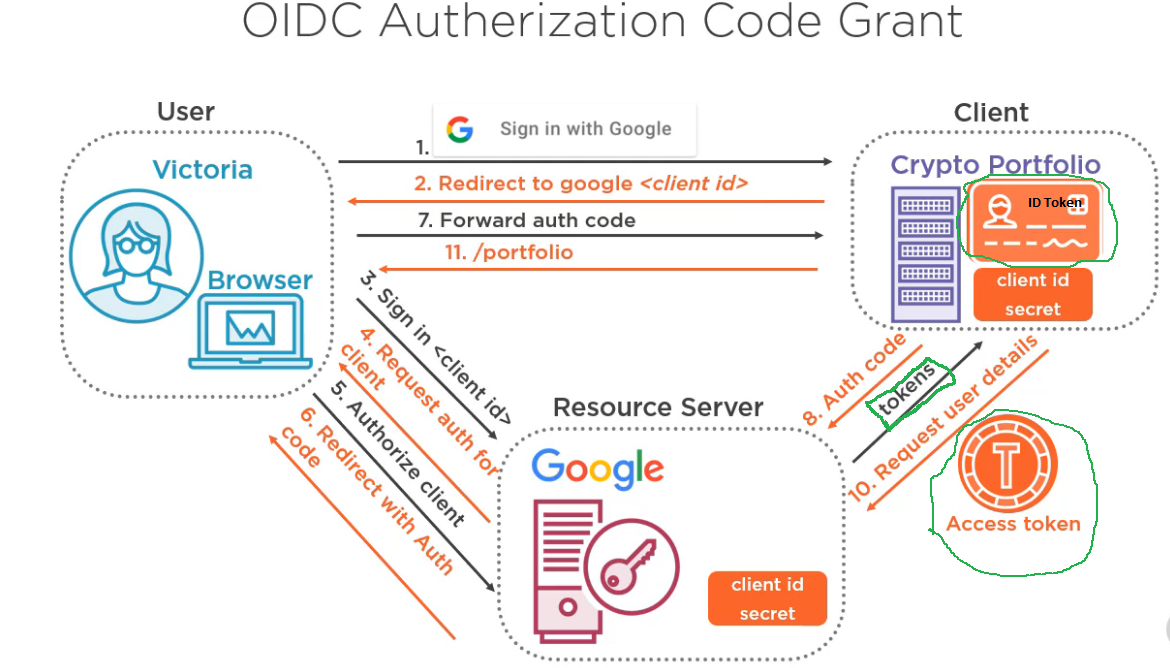
Let’s check the difference in code flow between ODIC and OAuth2

ODIC is built on top of OAuth2 so which include all the actors of OAuth2 (Client, Resource Server, Authorization Server, Resource Owner)



Facebook uses its own implementation of OAuth2 which is **Facebook Connect** which is very similar to ODIC.

**ODIC Authorization Code Grant:**



Now we Auth code is exchanged for ID Token and Access Token

**ID Token** is for the client to verify the authentication, which should be short crisp and have limited necessary details for authentication (FIN token), which can be easily transmitted in URL even in mobile client.

**Access Token** can be used against /userinfo end point to get additional details or claims about the resource owner.

**Refresh Token** to request for new Access Token after it expires