Salesforce – Identity Protocols

Dineshsing Girase
IT Dept., Mindtree Ltd
Philadelphia, USA

***Abstract*—There are various protocols available to implement identity solutions in Salesforce, like Security Assertion Markup Language (SAML), Open Authorization (OAuth 2.0), OpenId Connect etc. Salesforce identity connects users from Salesforce org to the external applications. It's very hard to identify which protocol to use when. This paper will help developers, architects with taking their decisions.**

Keywords—SAML, OAuth 2.0, OpenId, Protocol

I. INTRODUCTION

To confirm secure communication between the two systems, choose your authentication protocol wisely. Consider various configurations of the external systems, which you would like to connect with Salesforce. Go through plus/minus of below mentioned methodologies, which will help you decide right identity protocol to use.

II. PROTOCOLS

A. *Security Assertion Markup Language (SAML)*

- Security Assertion Markup Language (SAML) is an OASIS (Organization for the Advancement of Structured Information Standards) standards based protocol, used to setup Single Sign-On (SSO).
- Single Sign-On (SSO) lets users access other applications without logging in separately to each one.
- For example, users who already logged in to some app, then they can access their Salesforce org without logging in again.
- SAML is the protocol/standard data format used for exchanging the data between the two parties in SSO.
- SAML metadata is provided by the Identity Provider (IDP).
- The Identity Provider (IDP) is the one authenticating the user. The Service Provider (SP) is asking for the authenticated identity.
- When Salesforce acts as your Identity Provider, you can use a connected app to integrate a Service Provider with your org.
- All editions of Salesforce supports SAML based authentication.
- SAML is an XML-based protocol, which means that the packages of information being exchanged are written in XML.

B. *OAuth 2.0 (Open Authorization)*

OAuth 2.0 (Open Authorization) is an open protocol used to allow secure data sharing between applications.
Authorization = Do you have access rights to the resource?

OAuth 2.0 Components are

- **Roles (the Actors)**
  - Resource owner (End user/yourself)
  - Resource server (API server)
  - Client (application)
  - Authorization server (Token issuer)
- **Tokens and Endpoints**
  - Access Token
  - Refresh Token
  - The two main endpoints are the authorize endpoint and the token endpoint.
- **Scopes and Consent**
  - Scope is a parameter used to limit the rights of the access token
- **Grants**
  - Grant is a credential presenting the resource owner (me) my authorization used by the client to obtain an access token.
  - OAuth 2 specifies 4 grant types
    - Authorization Code Grant
    - Implicit Grant
    - Resource Owner Password Credentials Grant
    - Client Credentials Grant

C. *OpenID Connect*

- OpenID Connect is a protocol based on OAuth 2.0 that sends identity information from one service to another.
- You might have come across a prompt when you tried installing new app, like "Log in with your Google account"? That app is using the OpenID Connect protocol. When you sign in with Google, you're not creating an account and another password. Only Google holds that information. The app developer used the OpenID Connect protocol to enable social sign-on. In this case, Google is the identity provider.
- Open ID Connect is a protocol that enables SSO between two services. Unlike SAML, it adds an authentication layer on top of OAuth 2.0 to enable secure exchange of ID tokens that contain user information alongside OAuth access tokens.
- You cannot integrate identity providers with salesforce using connected apps. You would need an identity protocol in the middle.
- Advantages for business would be to automate or simplify the user creation process, reduce helpdesk interactions and provide a reliable source of user details.

III. GRANTS

A. Authorization Code Grant

- Use when the client is a web server
- You can obtain a long lived access token, as it can be renewed with a refresh token
- This is safer, because the access token is not passed on the client side
- In the below scenario, for example, Resource owner is you self, Client is any app on your device that you are using, Resource server is Google and Authentication server is Google.

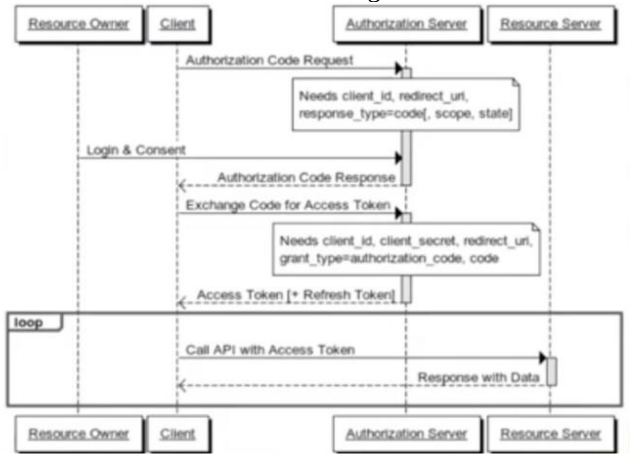
Resource Owner
Client
Authorization Server
Resource Server
Authorization Code Request
Needs client_id, redirect_uri, response_type=code[, scope, state]
Login & Consent
Authorization Code Response
Exchange Code for Access Token
Needs client_id, client_secret, redirect_uri, grant_type=authorization_code, code
Access Token [+ Refresh Token]
loop
Call API with Access Token
Response with Data

Fig.1 Flow - Authorization Code Grant

- You only see the access token, you never actually see-send the credentials for login. Hence this is much safer even if it's compromised.

B. Implicit Grant

- This is not recommended to use.
- It is typically used when the client is running a browser using a scripting language such as JavaScript.
- This grant doesn't issue the refresh token.
- Since it happens at browser side, there is no safe place to store the refresh token.

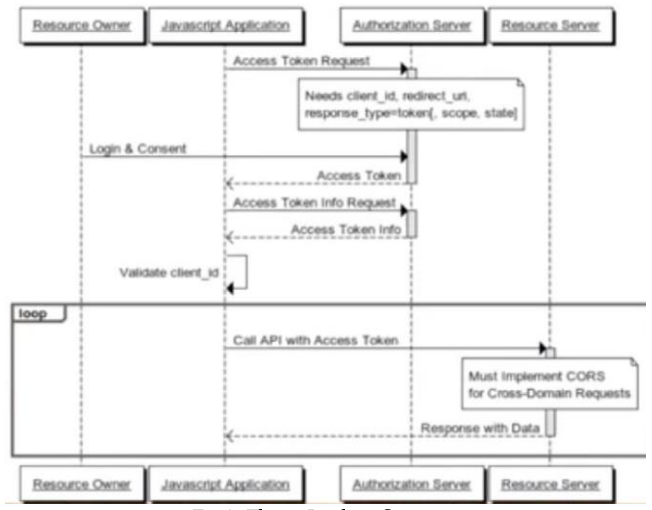

Fig.2. Flow - Implicit Grant

- In the above scenario, consider client as a Angular JS application, you are the resource owner, Authorization server and resource server is Facebook.
- When the access token is requested, and the resource owner gives the credentials, the authorization server redirects you back to the client application with the access token in the URL.
- The access token can now be used by the website and it can query the API putting that access token at the back of the API query as the parameter.
- So, in this, the access token is sent through HTTP packets which means anybody can intercept them. The people who can get the access token can use it to retrieve information from the resource server (Facebook in this case)

C. Resource Owner Password Credentials Grant

- With this type of authentication, the credentials and thus the password are sent to the client and then sent to the authorization server.
- This is good only when there is absolute trust between the two entities i.e., the client and the authorization server
- It is mainly used when the client has developed the same authority as the authorization server. For example, I want to access an app developed by salesforce and it is asking for my salesforce credentials to authenticate using the salesforce server to grant access, which is totally fine since there is implicit trust between the client and the authorization server.

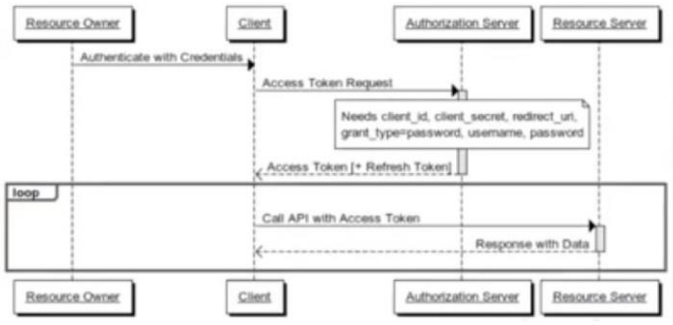
Resource Owner
Client
Authorization Server
Resource Server
Authenticate with Credentials
Access Token Request
Needs client_id, client_secret, redirect_uri, grant_type=password, username, password
Access Token [+ Refresh Token]
loop
Call API with Access Token
Response with Data

Fig.3. Flow – Resource Owner Password Credentials Grant

D. Client Credentials Grant

- This is one of the simplest of the OAuth 2.0 grants
- This is usable only for machine to machine authentication where a specific user's permission to access data is not required.
- This type of authorization is used when the client is himself the resource owner. There is no authorization to obtain from the end-user.
- The machine that is requesting access has its own account/access control that gives the needed access instead of requesting the user for access.
- Here the end user does not have to give its authorization for accessing the resource server, so the client's credential grant is used and not the resource owner's credential grant.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 07, July-2021

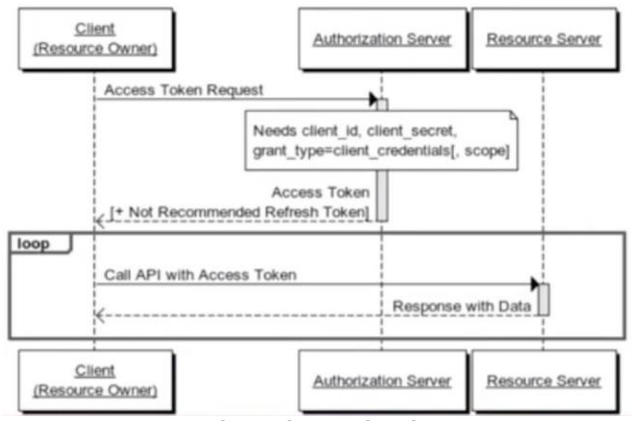
Client (Resource Owner)
Authorization Server
Resource Server
Access Token Request
Needs client_id, client_secret, grant_type=client_credentials[, scope]
Access Token
[+ Not Recommended Refresh Token]
loop
Call API with Access Token
Response with Data

Fig.4. Flow – Client Credentials Grant

IV. CONCLUSION - WHICH ONE TO CHOOSE

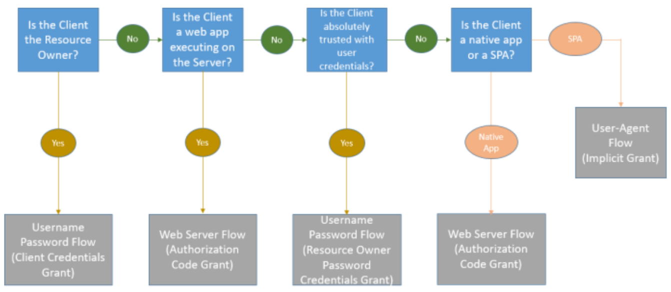
Is the Client the Resource Owner?
No
Is the Client a web app executing on the Server?
No
Is the Client absolutely trusted with user credentials?
No
Is the Client a native app or a SPA?
SPA
Yes
Yes
Yes
Native App
User-Agent Flow (Implicit Grant)
Username Password Flow (Client Credentials Grant)
Web Server Flow (Authorization Code Grant)
Username Password Flow (Resource Owner Password Credentials Grant)
Web Server Flow (Authorization Code Grant)

Fig.5. Make a decision

REFERENCES

[1] Learn the Language of Identity https://trailhead.salesforce.com/en/content/learn/modules/identity_basics/identity_basics_protocols

[2] How to Use Identity https://developer.salesforce.com/docs/atlas.en-us.identityImplGuide.meta/identityImplGuide/identity_how_to.htm

[3] Choose an Authentication Protocol https://help.salesforce.com/articleView?id=sf.named_credentials_auth_protocols.htm&type=5

[4] Identity Implementation Guide https://developer.salesforce.com/docs/atlas.en-us.identityImplGuide.meta/identityImplGuide/identity_overview.htm