Proof of Concept- Implementation of oAuth2 with Ping Federate

oAuth2 is not a new concept in the world of application security specially when comes to delegation of identity validation to a 3rd party. Many social applications today make use of well-known identity providers such as Facebook, Microsoft (Live.com) or Google to authenticate users where user provides credential to their known and trusted identity (ID) provider(s) (and not to the application such as Candy Crush Saga!). It works well for social applications as all parties (users, applications and service providers) trust the ID provider. Unfortunately, corporations can't trust or rely on public ID providers to authenticate users, applications and internal web api's for various reasons including strategic and core competencies. Enterprise must have complete authority (governance) over the Identity (Trust) provider regardless of ID provider sits on premise or in the cloud.

Okay, we are convinced that there are need for Federated Identity Provider trusted by users (customers and internal employees) and enterprise applications (any flavor). Companies can build such ID provider or they can buy the product available in the market. In my case, we are already using Ping Federate (on premise) as Identity Provider and Service Provider exchanging SAML 2.0 token with partners. So, why not leveraging Ping Federate infrastructure and simply add the oAuth2 capability?

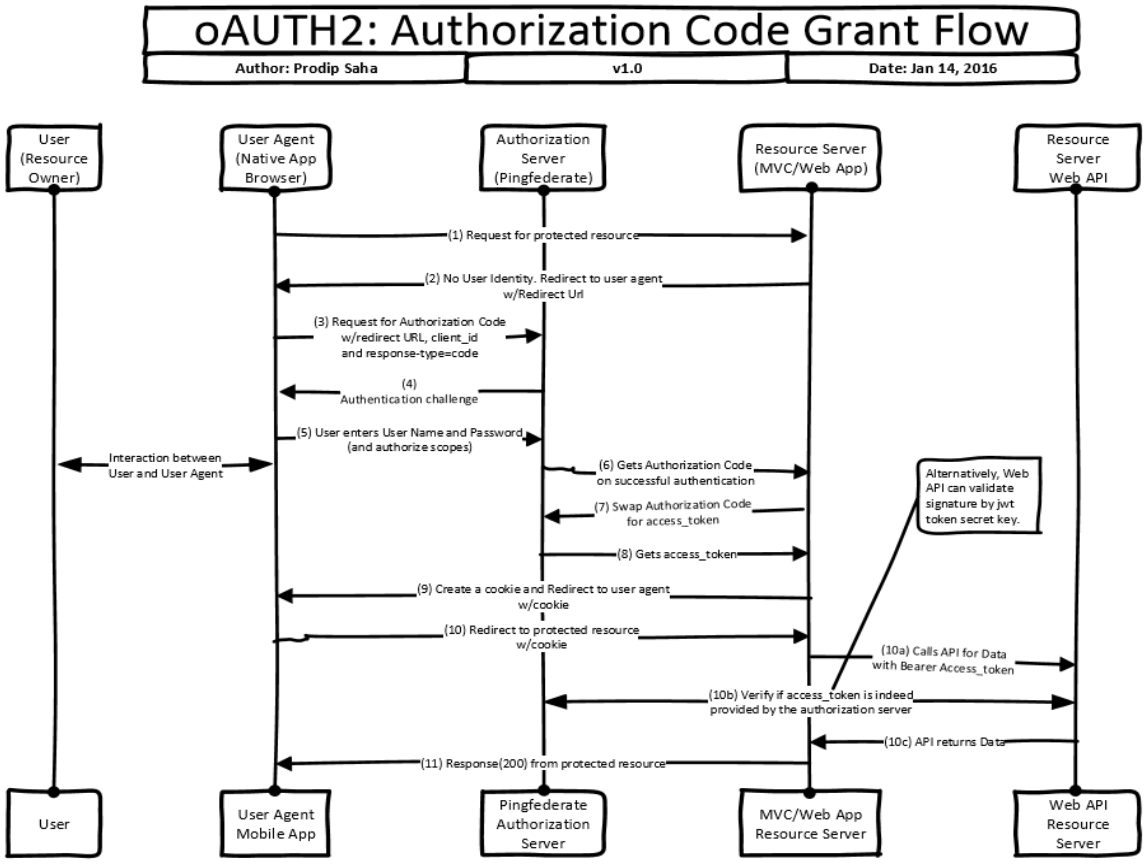
So, I am on a journey to prove an already proven concept of oAuth2! Ping Federate is all familiar to me as I have been working with Ping Federate for a while and I was looking for a jump start! In my POC scope, my objective was to authenticate an end user at ASP.Net MVC application (w/Ajax) and MVC application must use user's oAuth2 token to interact with a WebApi application.

The best place to start is - OAuth2 Developers Guide - Ping Identity. This document provided me a good understanding of different actors and oAuth2 flows. Below are the main actors-

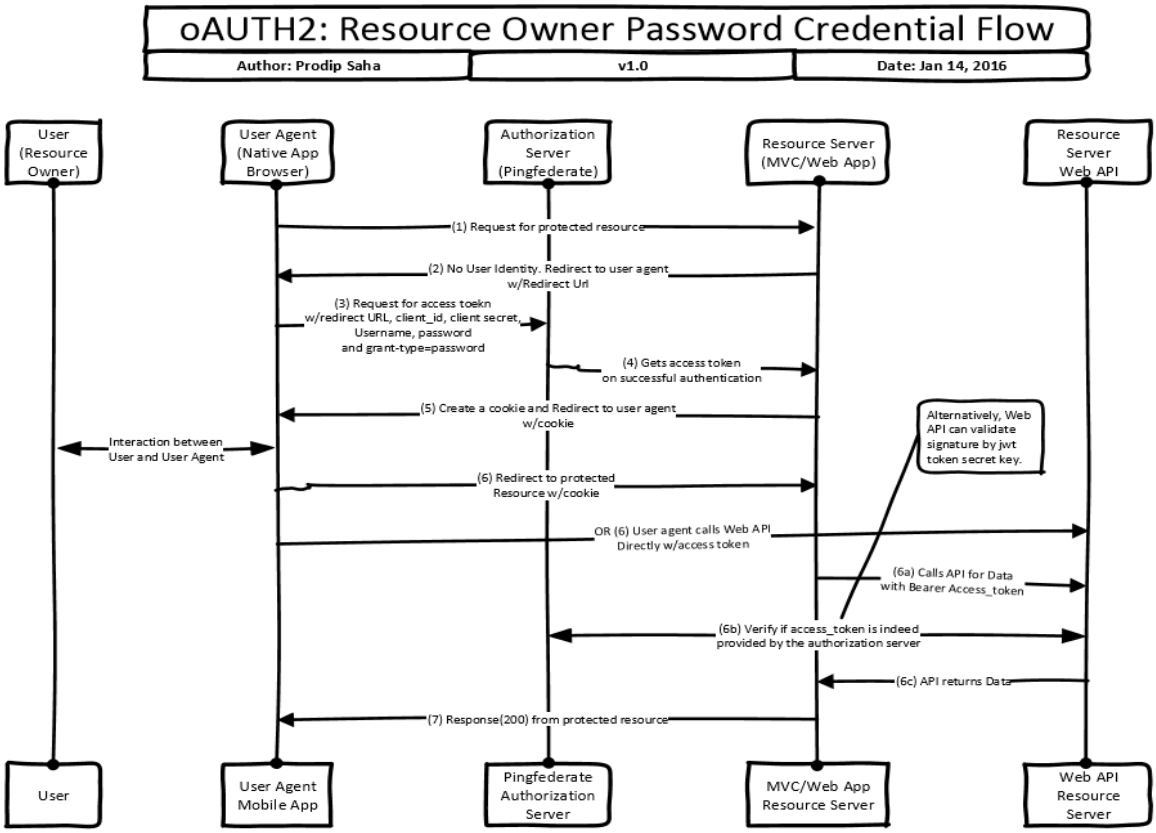
Actor	Responsibility
User or Resource Owner	The actual end user, responsible for authentication and to provide consent to share their resources with the requesting client.
User Agent	The user's browser. Used for redirect-based flows where the user must authenticate and optionally provide consent to share their resources.
Client	The client application that is requesting an access token on behalf of the end user.
Authorization Server (AS)	The PingFederate server that authenticates the user and/or client, issues access tokens and tracks the access tokens throughout their lifetime.
Resource Server (RS)	The target application or API that provides the requested resources. This actor will validate an access token to provide authorization for the action.

There are two oAuth2 flows that I wanted to pay special attention- ***Authorization Code Grant*** and ***Resource Owner Password Credentials***. In addition, I wanted to test ***Client Credential*** flow as it is suitable for service to service authentication/authorization because there is no end user involved in the chain.

Authorization Grant Flow works for most of the Web and Mobile applications including JavaScripts clients. It's a two-step process- get the authorization code by exchanging username and password and then client would swap the authorization code to get access token. In this flow user never disclose username and password to client application and credentials are exchanged with identity provider directly to get the authorization code.



Resource Owner Password Credentials flow is to provide backward compatibility to existing web applications. This is true when you have to allow the end users login to an existing web form. User provides username/password and client combines username, password with client_id and client_secret to get the access token directly from identity provider. As you can see, user's credentials are known to client application and it leaves the possibility of misuse of user's credentials. Nonetheless, this flow can be used where client (web app) is fully governed by the same company owns the identity provider.



Enough of reading the OAuth2 Developers Guide - Ping Identity! Let's configure oAuth2 in Ping Federate following the documentations. There are few configurations that you do need to complete before you can get started with oAuth and you can follow oAuth Configuration admin guide to configure them. Thanks to Curtis Muir (RSA at Ping Identity) who got me jump started on the configurations. Following steps are required for Authorization Grant Flow (along with the documentation links).

- Enabling the OAuth AS
- Authorization Server Settings
- Access Token Management
- Client Management
- IdP Adapter Mapping for OAuth
- Access Token Mapping

OpenId (Configuring OpenID Connect Policies) is where you define the policy and scopes. Scopes are important in the world of oAuth because client will get response if user grants the scope. You may be already familiar with scopes if you used oAuth login to Facebook redirected by an application where you (user) have to grant permission to read- name, email, phone, location, etc. Name, email, phone, etc. in this case are considered scopes. Also, OpenId is required if you are trying to validate an access token with trust (Ping Federate) just to double check that client did not tamper with the token. Keep in mind

that a client (usually an application) can decrypt a token, rebuild and re-sign it provided the client has access to secret. It's important for service providers (i.e. WebApi's down the chain) to check the integrity of the access token with the ID provider- using OpenId. Resource-Owner Credentials Mapping configuration is required if you are trying to use Resource Owner Password Credentials flow.

Well, oAuth2 configuration is ready but how do we test the flows? We don't want to use MVC Web App at this moment and until we have a way to test the flows. Good news is- Ping Federate provides Playground sample (PingFederate OAuth 2 Playground 3.3) that you can download and setup under IIS. You will need to adjust some settings based on your environment but these simple html sample works like a magic!

Okay, so much fun with the playground! Playground html pages are wonderful to conduct sanity tests before moving to Asp.Net MVC and/or WebApi. This is the time you will discover issues and you would resolve them.

ASP.NET MVC with Ping Federate oAuth2 Provider

There is no need to reinvent especially when you are trying to conduct a POC. Owin.Security.Providers.PingFederate can get you up to speed. I used this Ping Federate OWIN provider module along with WorkingClient with some modifications.

Ping Federate ships the product with a number of adapters and validators but it did not have the one I needed. I needed to validate user's credential in SQL database but the credentials are encrypted in SQL (one-way hashed and salted with custom algorithms). So, I had to create a new SQL Password Validator using java codes. I am not a seasoned java developer but it did not take me long to create a new password validator following the SDK Developer's Guide documentation and sample codes (pf-pcv-sqlstoredprocedure). Yes, I needed to install Apache Commons and set the path in environment variable. Of course, I needed Java JDK but I always have it for other purpose in my environment!

Basically, Working Client would redirect the user to Ping Federate credential validator login page to get authorization code. Once WorkingClient has the authorization code, it can swap the code for access token. MVC application would use the access token, validate it and create an authentication cookie which is sent to the browser. All subsequent calls are validated by the issued cookie. It's the Ping Federate provider that would help you validate access token.

WebApi

So far so good and we have successfully authenticated the browser based client and user with our identity provider (Ping Federate) and MVC application is able to validate the user's access token issued by the trust (Ping Federate). We are covered for point-to-point authentication and authorization. MVC application is the presentation tier and it needs to communicate with Web Api's to get the data. We have to ensure that requests initiated by JavaScript client or requests initiated by MVC client are validated at the Web Api layer. The task of validation at the WebApi became easy due to presence of

common trust. JavaScript or MVC client can pass the access token to WebApi and WebApi can verify the token with trust (Ping Federate) without knowing the client secret. It does not matter who calls whom, the validation of token became so easy because they all trust Ping Federate. What do we gain at the end? We are able to enforce end-to-end security at the disconnected applications as long as they toss around the same access token.

In my use case, I created an Api application using ASP.Net MVC WebApi empty template. I used JSON Web Token Handler to validate JWT token. You can simply override the ValidateToken method overload that takes TokenValidationParameters. You decide what are the parameters that you would like to validate- signature (a must), audience(s), issuer(s), lifetime, etc. You must provide the client secret to validate the signature.

We are not going to talk much on Resource Owner Password Credentials since it is only good fit for legacy client.

JWT Debugging

Jwt (JSON Web Token) is industry standard and it was my obvious choice. You can use https://jwt.io/ to debug jwt tokens out of the band. Just provide the client secret to validate the signature.

Example of valid Jwt (with client secret being “secret”): Signature would be invalid if you change the encoded token or change the client secret.