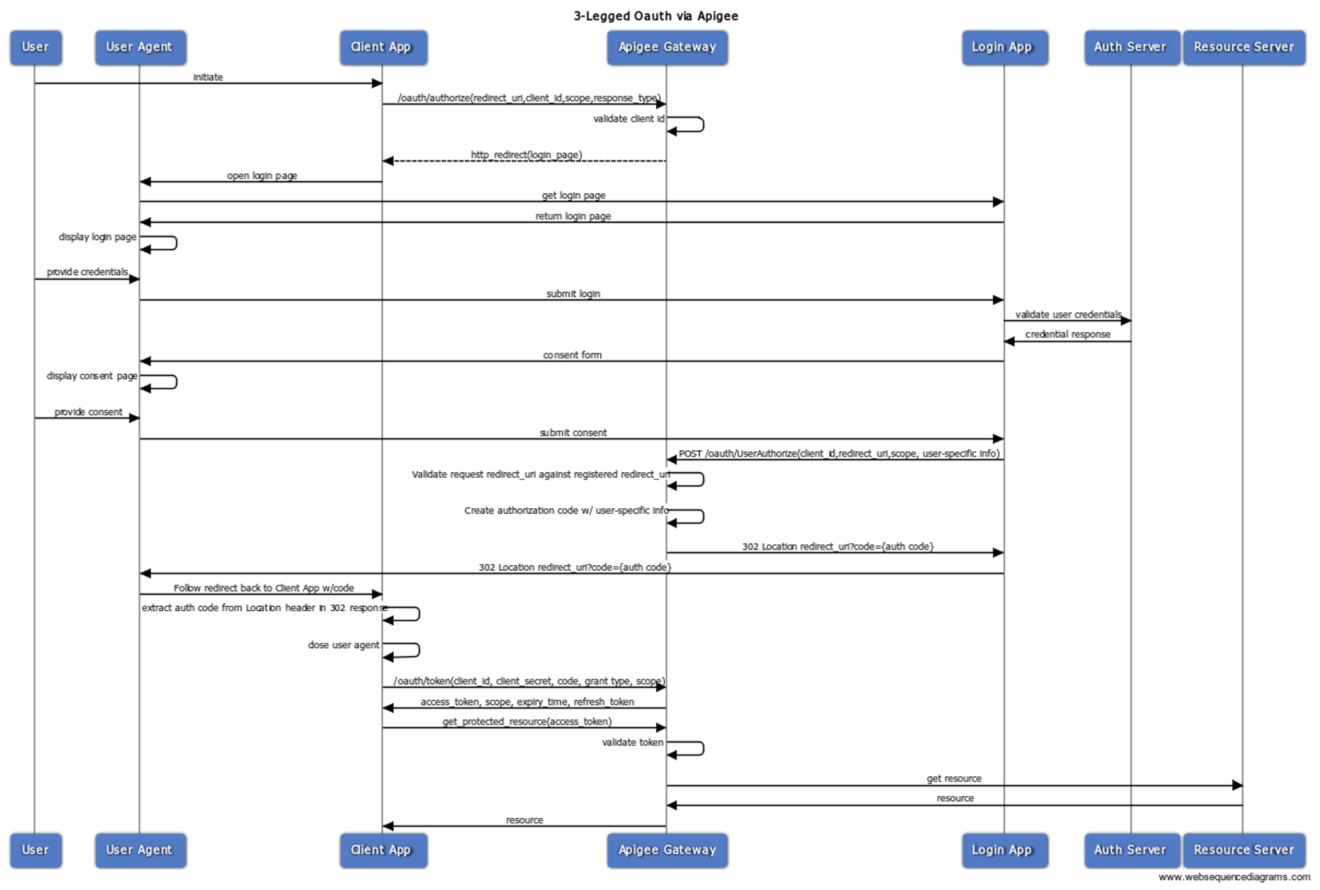
Security Architecture and Engineering (SAE): OAuth and RESTful Service Authentication - Part 2: Authorization Code Grant

Posted by Steven Nakhla Sep 26, 2017

In my last blog post, I discussed how to perform service account authentication through Apigee using the Client Credentials grant type. When we were finished our exercise, a service account was able to provide a username and password to Apigee, and in return obtain an OAuth access token that could be presented in subsequent Apigee service requests. Today, we'll discuss how to perform user account authentication via OAuth using the *Authorization Code* grant type.

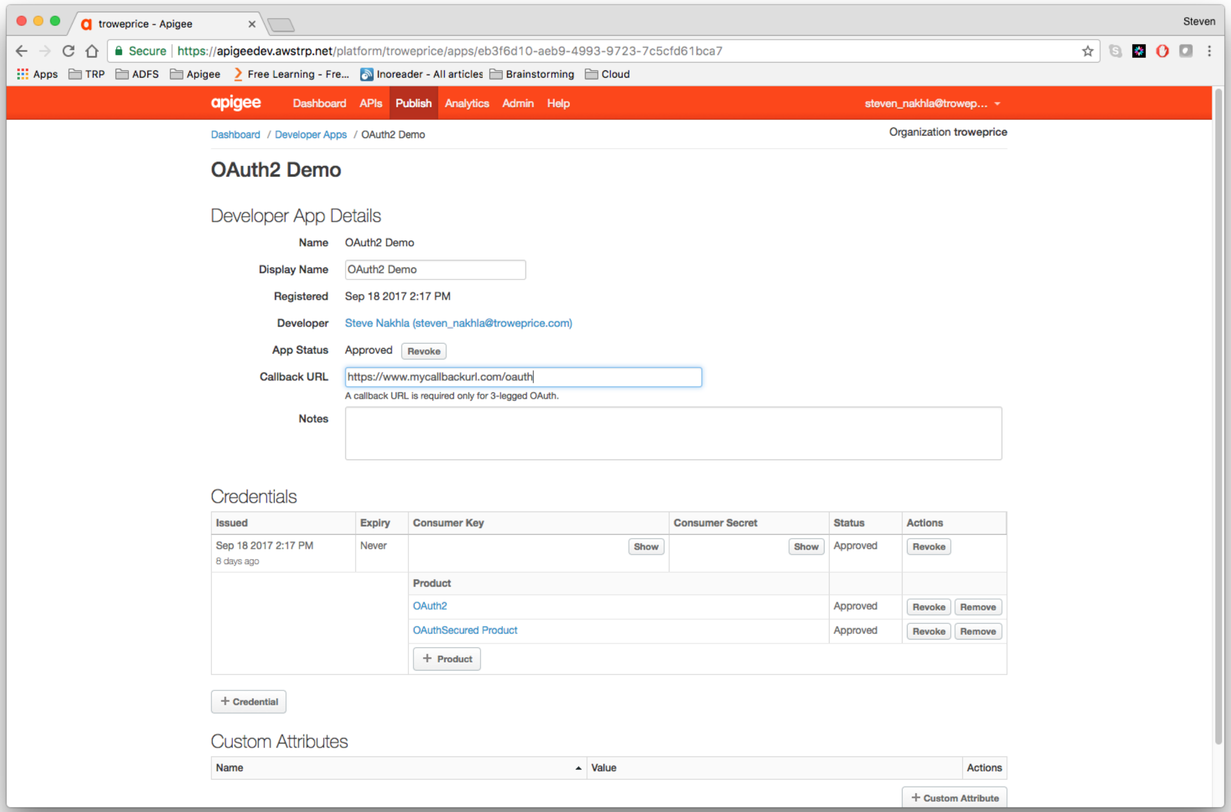
Apigee is well-suited to act as a reverse proxy for RESTful service calls. It does not, however, perform the role of a user-facing authentication gateway. As a result, we have decided to leverage Microsoft's Active Directory Federation Services (ADFS) as our OAuth authorization server. ADFS is already in use across the firm in a variety of capacities. It is used for authenticating to Office 365, as well as to the Amazon Web Services console. Because it is tied to our existing Active Directory infrastructure, it is well-suited to provide user authentication. ADFS already supported SAML for identity federation. The latest version also includes support for OAuth. Apigee has been configured to act as an ADFS client, with its own client ID and client secret, used to establish trust between Apigee and ADFS. Because of this, authentication calls that begin through Apigee do not need to worry about integrating with yet another endpoint. Apigee will broker the authentication ceremony, making it much easier to integrate with your applications and services.


3-Legged Oauth via Apigee
User
User Agent
Client App
Apigee Gateway
Login App
Auth Server
Resource Server
initiate
/oauth/authorize(redirect_uri,client_id,scope,response_type)
validate client id
http_redirect(login_page)
open login page
get login page
return login page
display login page
provide credentials
submit login
validate user credentials
credential response
consent form
display consent page
provide consent
submit consent
POST /oauth/UserAuthorize(client_id,redirect_uri,scope, user-specific info)
Validate request redirect_uri against registered redirect_uri
Create authorization code w/ user-specific info
302 Location redirect_uri?code={auth code}
302 Location redirect_uri?code={auth code}
Follow redirect back to Client App w/code
extract auth code from Location header in 302 response
close user agent
/oauth/token(client_id, client_secret, code, grant type, scope)
access_token, scope, expiry_time, refresh_token
get_protected_resource(access_token)
validate token
get resource
resource
resource
User
User Agent
Client App
Apigee Gateway
Login App
Auth Server
Resource Server
www.websequencediagrams.com

The diagram above shows a very typical user-facing OAuth interaction. While our model will not match this example 100em, it does give a sense of the overall architecture of the Apigee/ADFS integration. As shown above, the client web application recognizes that a user must be authenticated to access *Resource X*. The user is redirect to the Apigee authorize URL, providing information about the calling application. Apigee will then redirect the user via an HTTP 302 to the ADFS login page. It is important to note that this step ensures that user credentials are only provided to ADFS, and are not accessible to Apigee or any other services. The user will enter their Active Directory credentials and, after being authenticated by ADFS, be redirected back to the client application's *OAuth Callback URL*. This URL is established in the Developer App configuration. Apigee will *ONLY* redirect to this URL. (**NOTE**: The example image shows the typical 3-Legged OAuth flow, where a consent page is shown to the user prior to redirecting to Apigee. This is the type of interaction you would see if logging into a site using your Facebook account, for example. Because TRP is both the identity provider and service provider, the consent page is not relevant to our environment.) The redirect to the client application includes an *authorization code*. This code indicates that the user has successfully authenticated to the identity provider. Upon obtaining this authorization code, the client application can exchange it for an OAuth access token by performing an HTTP POST to the Apigee token endpoint. This call ***MUST*** take place on the backend server, as it includes the Consumer Secret for the application. It must not be performed via the client browser directly, as doing so would expose the consumer secret value. Once Apigee validates the Consumer Key and Consumer Secret values, it will make a call to an ADFS endpoint to validate that the authorization code is valid. If so, an OAuth *access token* is returned by ADFS, and subsequently returned back to the client application web server.

Looking at the description of the OAuth Authorization Code grant type, it is easy to become overwhelmed and think that it is a very difficult integration. However, it is important to note that much of the heavy lifting is performed on the Apigee side, so there is no need to implement it in your applications. Our demo will show that integrating OAuth authentication into your application is not as difficult as it may seem.

To begin adapting our Developer App for OAuth authentication, first go to the Developer Apps link under the Publish menu within the Apigee Edge console. Locate and click on our OAuth2 Demo application to view the application details. Initially, you will notice that our *Callback URL* parameter is empty. This is because this parameter is not necessary when authenticating service accounts using the Client Credentials grant type. To take advantage of the Authorization Code grant type, however, we must provide this value. As mentioned earlier, it is imperative that this callback URL not expose any sensitive information to the user, such as the Consumer Secret value. To set the Callback URL value, click the *Edit* button at the top right-hand corner. Enter your Callback URL value and click *Save* at the bottom of the page.



OAuth2 Demo

Developer App Details

Name	OAuth2 Demo
Display Name	OAuth2 Demo
Registered	Sep 18 2017 2:17 PM
Developer	Steve Nakhla (steven_nakhla@troweprice.com)
App Status	Approved Revoke
Callback URL	https://www.mycallbackurl.com/oauth A callback URL is required only for 3-legged OAuth.
Notes	

Credentials

Issued	Expiry	Consumer Key	Consumer Secret	Status	Actions
Sep 18 2017 2:17 PM 8 days ago	Never	Show	Show	Approved	Revoke
		Product			
		OAuth2		Approved	Revoke Remove
		OAuthSecured Product		Approved	Revoke Remove
		+ Product			

+ Credential

Custom Attributes

Name	Value	Actions

+ Custom Attribute

Now that this is done, we can begin implementing the backend endpoint of your Callback URL that will be used to exchange the OAuth authorization code for an OAuth access token. The authorization code will be provided as the code query parameter in the call to your Callback URL. The specific implementation details will depend upon what language and platform you will be using. The commands for .Net differ from those used for Java and Spring Security, for example. So, the specific implementation of this functionality is left as an exercise to the reader. As a high-level description of this functionality, the curl command below shows the format of the HTTP POST to the Apigee token endpoint.

```
curl -kvi -X POST -u <CONSUMER KEY>:<CONSUMER SECRET> -H "Content-Type: application/x-www-
form-urlencoded" -d \
"grant_type=code&client_id=<CONSUMER KEY>&code=<AUTHORIZATION CODE>" https://
apidev.awstrp.net/oauth2/token
```

You will notice that our HTTP POST specifies the following form parameters:

- grant_type: This will always be "code" for the Authorization Code grant type
- client_id: The Consumer Key for your Developer App
- code: The Authorization Code received by your Callback URL endpoint

It is also important to note that the HTTP POST to the Apigee token endpoint must also include your *Consumer Key* and *Consumer Secret* as Basic Authentication credentials. This is done to ensure that the Authorization Code is being submitted by an authorized application known to Apigee.

Once you perform the authorization code/access token exchange with Apigee, you will receive a JSON payload that resembles the following:

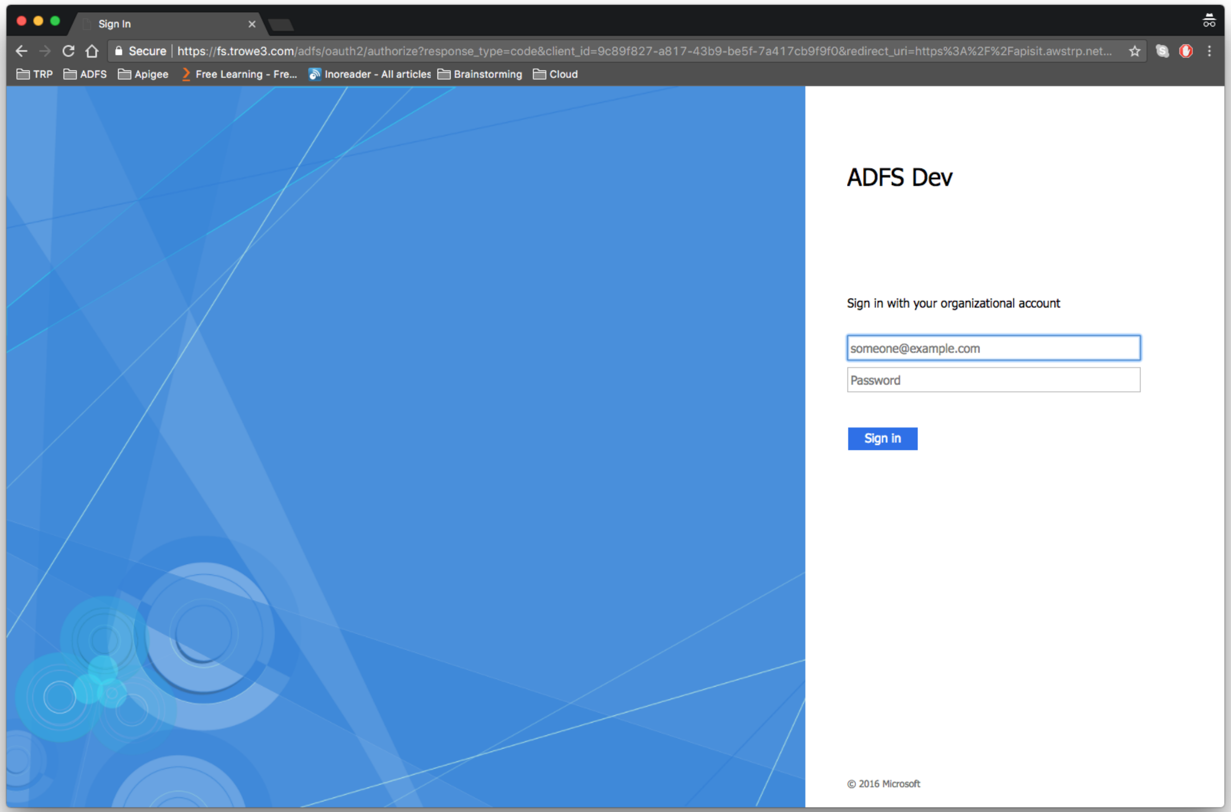
```
{
  "refresh_token_expires_in" : "0",
  "refresh_token_status" : "approved",
  "api_product_list" : "[OAuth2, OAuthSecured-Product]",
  "organization_name" : "troweprice",
  "developer.email" : "steven_nakhla@troweprice.com",
  "token_type" : "BearerToken",
  "issued_at" : "1505399022363",
  "client_id" : "<CONSUMER KEY>",
  "access_token" : "<OAUTH ACCESS TOKEN>",
  "refresh_token" : "<OAUTH REFRESH TOKEN>",
  "application_name" : "d7b080d2-9519-420b-a7c6-75e0a4139454",
  "scope" : "",
  "refresh_token_issued_at" : "1505399022363",
  "expires_in" : "35999",
  "refresh_count" : "0",
  "status" : "approved"
}
```

The access_token value is our OAuth access token that can be used in subsequent RESTful service calls through Apigee. Now that our Developer App and backend web application configuration are complete, let's walk through the actual authentication process and see if our integration was successful.

To begin the authentication process, the Apigee OAuth authorization endpoint should be used. This service endpoint is designed both for convenience, as well as to establish a trust between Apigee and ADFS. In DEV, the endpoint would look like this:

https://apidev.awstrp.net/oauth2/authorize?apikey=<YOUR CONSUMER KEY>&response_type=code

Accessing this URL through your web browser will redirect you to the ADFS login page with the parameters necessary to establish the trust between Apigee and ADFS. The login page will look like this:



https://fs.trowe3.com/adfs/oauth2/authorize?response_type=code&client_id=9c89f827-a817-43b9-be5f-7a417cb9f9f0&redirect_uri=https%3A%2F%2Fapisit.awstrp.net...

ADFS Dev

Sign in with your organizational account

someone@example.com

Password

© 2016 Microsoft

Enter your TRP e-mail address and Active Directory password into the login form, and click *Sign In*. At this point, ADFS will authenticate your credentials against Active Directory, and redirect you to your application's Callback URL with the OAuth Authorization Code. If you are viewing the logging of your Callback URL, you should see the call come in from the browser after the 302 redirect. At this point, your Callback URL endpoint will perform the HTTP POST to the Apigee token endpoint to obtain an OAuth Access Token. If successful, the JSON payload returned by this call should look like this:

```
{
  "refresh_token_expires_in" : "0",
  "refresh_token_status" : "approved",
  "api_product_list" : "[OAuth2, OAuthSecured-Product]",
  "organization_name" : "troweprice",
  "developer.email" : "steven_nakhla@troweprice.com",
  "token_type" : "BearerToken",
  "issued_at" : "1505399022363",
  "client_id" : "<CONSUMER KEY>",
  "access_token" : "<OAUTH ACCESS TOKEN>",
  "refresh_token" : "<OAUTH REFRESH TOKEN>",
  "application_name" : "d7b080d2-9519-420b-a7c6-75e0a4139454",
  "scope" : "",
  "refresh_token_issued_at" : "1505399022363",
  "expires_in" : "35999",
  "refresh_count" : "0",
  "status" : "approved"
}
```

Success! Our payload has an access_token value, indicating that our OAuth authentication was successful. We can now return this access token to the caller and use it for subsequent RESTful service calls through Apigee, similar to how we did with the Client Credentials grant.

We have now demonstrated how service accounts can perform OAuth authentication using the Client Credentials grant type, as well as how users of web applications can authenticate using the Authorization Code grant type. In my next post, I will discuss how to integrate OAuth authentication into native iOS mobile applications. As always, if you have any questions regarding OAuth authentication through Apigee, please do not hesitate to reach out to me!

Steven Nakhla

Enterprise Security
Security Architecture and Engineering (SAE)
Middleware & Engineering
2113 Views Tags: rest, authentication, oauth, cloud security, enterprise security, apigee, rest api, adfs

Dennis Smith
Aug 9, 2019 2:30 PM
Is it possible to authenticate with AD and retrieve an Authorization code without having an api key and callback URL? Or is that always done through a redirect.

In other words, can I authenticate with my creds through Postman hitting https://api.dev.awstrp.net/oauth2/authorize?

Thinking of a case where maybe I have service I need to develop and test locally and wire in internal entitlement checks before any client app is provisioned.

Maybe I'm thinking about this wrong.

Nikhil Mujumdar
Dec 7, 2018 1:03 PM
The document needs to be updated with new URL's for Apigee developer portal.
Also on entering https://apidev.awstrp.net/oauth2/authorize?apikey=<YOUR CONSUMER KEY>&response_type=code it just shows the standard OS authentication box and not the ADFS login page.
You still need to enter your email and ad password like before.

Mathieu Lorentz
Feb 3, 2018 5:31 PM
It's no longer possible to create a developer app from apigeedev. Can you update the doc?
Thanks.

Mark Gogel
Oct 31, 2017 9:59 AM
How am I just seeing this? Great blog post pt2 Steve!