Security Architecture and Engineering (SAE): The Future of RESTful Service Authentication - Introduction

Posted by Steven Nakhla Sep 13, 2017

As part of T. Rowe Price's migration toward cloud computing technologies, we have begun taking a long, hard look at how services are developed. While we have long had RESTful services here at TRP, they have been implemented in a very inelegant and inefficient way, due to limitations in our technology stack. As a result, the firm has begun looking at ways to modernize our services infrastructure and design.

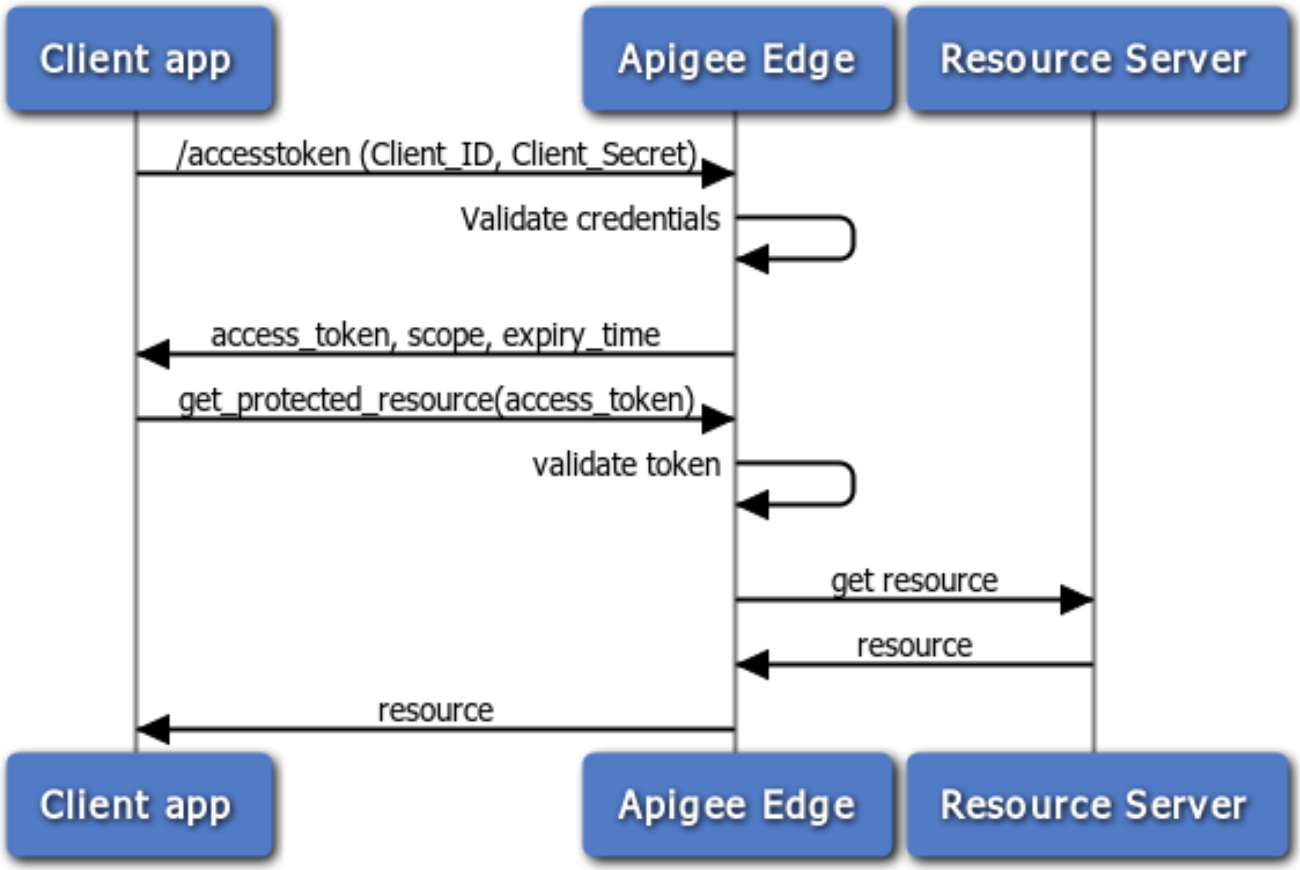
In the "old way" of doing RESTful services, the services themselves were consolidated (and externalized, where appropriate) using Tivoli Access Manager (now iSAM), a reverse proxy for HTTP connections. The familiar "TAM Junction" was mapped to a set of backend servers that were responsible for serving up the RESTful endpoint. While this setup served its purpose, it had many shortcomings. TAM/iSAM is mostly designed as a reverse proxy for HTML-based applications, not RESTful services. As such, it tends to "swallow" exceptions and error codes that would normally be returned to the client. Additionally, authentication was always tricky when dealing with RESTful services. TAM hosts its own HTML-based login page, and normally expects that users will authenticate there. As TRP began developing mobile applications for iOS -- applications that relied heavily on RESTful services for functionality -- it became evident that this would not work. So, the MAX service was developed. MAX was a RESTful service designed to handle user authentication. TAM would delegate authentication to MAX, and MAX would perform the task of validating user credentials against Active Directory.

As you can see, the complexity and compromises needed to mitigate the shortcomings of our "old way" prevented us from following industry best-practices. As such, a new technology stack has been implemented to address our needs for RESTful services. The Apigee API Gateway has become our new standard endpoint for RESTful services. Implemented by Middleware & Engineering, Apigee provides a rich workflow for developing and deploying RESTful services in the AWS cloud. With this new API gateway comes an opportunity to change how we are doing authentication for these new services.

OAuth is an open standard for delegated authorization, commonly used as a way for users of a website (Google, Facebook, etc.) to provide access to their data to 3rd-parties, without being forced to give up their login credentials. We have all seen sites that allow you to "Login with Facebook" or incorporate your Twitter feed into a mobile application. This is all accomplished thanks to OAuth. The Apigee API Gateway has native support for OAuth, and makes it simple to require OAuth authentication prior to accessing a protected API.

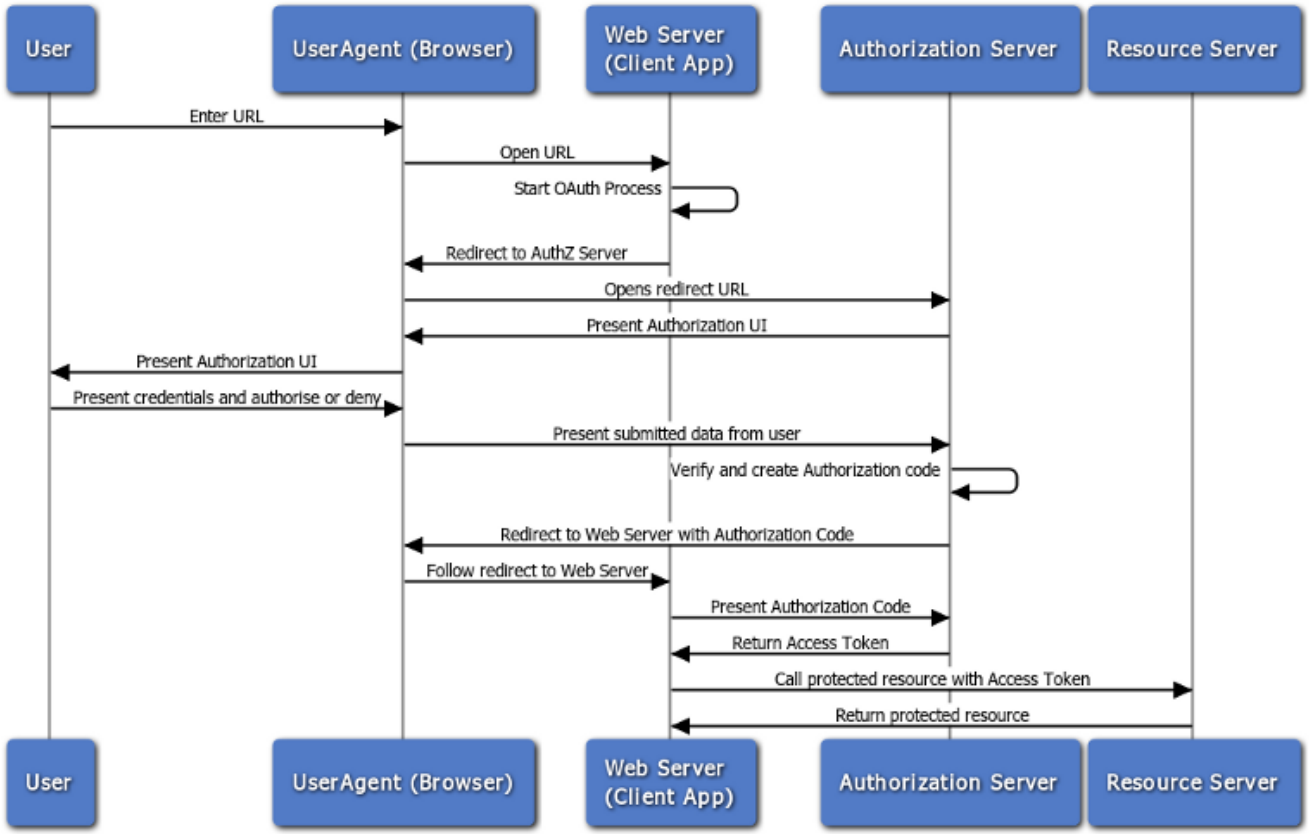
The OAuth standard supports several different types of "flows" or grant types, depending on your use case. Each grant type has a different interaction with the client and authorization server, based on its security requirements. The two most common grant types are the *Client Credentials (or Password) Grant* and the *Authorization Code Grant*.

With the *Client Credentials* grant type, the client is typically the resource owner themselves. That is, the caller is also the owner of the resource or data being accessed. This type of Grant is most commonly used for service-to-service interaction, such as batch jobs, cron jobs, etc. The diagram below shows a typical Client Credentials flow. You'll notice that the client (typically a service account) provide credentials for authentication for validation. Once validated, an OAuth access token is returned to the client. This access token is included in all subsequent RESTful service calls. The API Gateway (Apigee Adge) validates the token each time the client makes a call to ensure that they have properly authenticated.



The *Authorization Code* grant type is used for web applications executing on a server, as well as native mobile apps running on a device. In this grant type, the client receives an "authorization code" from the OAuth authorization server. The authorization code can then be exchanged via a server-side call (or native call in the case of mobile applications) for an OAuth access token. The diagram below shows a typical Authorization Code flow for web applications. While the interaction does look more complex, much of this complexity can be hidden via the Apigee API gateway, or through the use of 3rd party application frameworks. You'll see that the user authenticates directly agains the Authorization Server, rather than passing their credentials to an intermediary, such as an API Gateway. Once authenticated, the Authorization Server will return an

Authorization Code. The Web Server (or native application) can then call the Authorization Server directly and exchange the Authorization Code for an OAuth Access Token.



In order to implement OAuth authentication here at T. Rowe Price, Security Architecture and Engineering (SAE) has worked closely with both Middleware & Engineering and WIS to ensure that the necessary components of our technology stack are able to interoperate in a secure and efficient manner. Active Directory Federation Services (ADFS) is a technology from Microsoft that provides users with single sign-on support to systems and applications. Many of you may already be familiar with ADFS, as it is used to sign in to Office 365 or the Amazon Web Services Console. It is historically used to provide SAML-based login functionality, but recent versions have also added OAuth support. ADFS will be acting as our OAuth authorization server for ***associate-facing services***, and we will be integrating it with Apigee to provide seamless authentication and identity propagation functionality to all RESTful services. Enterprise Security will be implementing support for both the Client Credentials grant type, as well as the Authorization Code grant type, in order to support Service->Service, User->Service, and Mobile User->Service authentication.

Over the next few days, I will be posting more blog entries detailing how to implement OAuth functionality in your services and applications. In the meantime, should you have any questions about Apigee, Prasad Vepa should be your point of contact. If you have questions regarding OAuth or API security in general, please do not hesitate to reach out to me.

Steven Nakhla
762 Views Tags: security, rest, oauth, aws, security engineering, apigee, rest api

There are no comments on this post