**OAuth2 Token Exchange for the Enterprise API Gateway**

**API Security - Three Goals**

https://youtu.be/7VRTWGqdP54

Many within the IT group have access or knowledge of production credentials.

OAuth originated for the web and is designed for the web. It has since been adopted by Enterprises which have additional needs. To handle the second generation OAuth2 was written in a modular way with additional specifications [4]. These modular specification are combined to accomplish Enterprise use cases. The use case covered in this article is propagating the authenticated user’s security context thru an API Gateway to an API. This can be accomplished using the specification RFC 8693 “OAuth2 Token Exchange” [2]. In this article shows how our large enterprise used token exchange to secure our API Gateway and APIs and provide a full implementation with Auth0 that you can run yourself.

**Our enterprise identified three goals:**

* **The application cannot skip the API Gateway and authorize directly to the API.**
* **The API behind the API Gateway knows the authorized user’s context including their roles.**
* **The API Gateway cannot arbitrarily act as an authorized user.**

I encourage you to read “Identity Propagation in an API Gateway Architecture” by Robert Broeckelmann [3] for an in-depth discussion on why propagating a user’s context is advantageous and worth the extra work. This article will fully implement that extra work in Auth0.

**The application cannot skip the API Gateway and authorize directly to the API.**

This goal excludes passing forward an access token thru the API Gateway. Per the OAuth2 standards access tokens and id tokens have an audience and are meant to be used by the application the audience is registered to [1]. Sharing audiences (or application registrations) would quickly nullify the value of our security approvals and audits. If you have several small APIs that operate as a single unit OAuth2 likely isn’t necessary with modern cloud security options.

**The API knows the authorized user’s context**

Robert Broeckelmann explains wonderfully why knowing the originating authenticated user at each step increases the overall security of a system [3]. A key point is lessening the breadth of access of “service accounts”. The de-facto approach is to use “service accounts” that have full access and then limit that access through some sort of parameterization when calling other APIs. The user and their permissions are lost in translation. Not so when the authorized user’s context is passed forward. The entire execution chain can be limited by the rights of the authorized user. This ties in closely with the next goal.

**The API Gateway cannot arbitrarily act as an authorized user**

As stated, it is common for service accounts to have full access and limit their actions by parameterized API calls. This is reversed with token exchange. A service account now has no rights until it adopts the context of the authorized user that started the execution chain. In OAuth2 the context is contained is a signed token that expires. So, the service account cannot construct the signed token and the service account can only act as the user for a configurable period. This maintains a key benefit of OAuth2 through the entire execution chain.

This reduces the damage if a service account’s credential is compromised. This can be referred to as a “blast radius”. If a service account can act as any user the “blast radius” of it’s credentials are massive. However, if the service account cannot do anything without an authorized user at the start of the execution chain the “blast radius” is minimal. The responsibility stays on the user to maintain their credentials. This protects against internal attacks.

**Auth0**

Auth0 is used for this example as the secure token server but is not the only option. There are many implementations of OAuth2 with their own flavor. I use Auth0 because that is our vendor, it has a great free account option and a management API [5] that allowed me to write a PowerShell script that generates all the needed components. Other platforms can also do token exchange.

**PowerShell Script - Auth0 Management API**

Included is a PowerShell script ‘TokenExchangeAuth0.ps1’ that uses the Auth0 management API [5] to create everything needed to run the Token Flow discussed next. Even if you are not using Auth0 I recommend setting up the example in a free Auth0 account. There’s nothing like seeing it work! The components in Auth0 translate to other OAuth2 platforms.

**Token Flow**



1. Application – Authenticate the user using Authorization Code Flow
   1. Use Authorization Code Flow to login the user then get ‘Access Token A’. The details of Authorization Code Flow will not be covered here [7].
   2. Access Token A
      1. Contains the user id (subject) and their permissions (permissions).
      2. Used to authenticate the call to the API Gateway
2. Application - Call API Gateway endpoint
   1. Include ‘Access Token A’ in the header (Authorized: Bearer …)
   2. API Gateway authorizes the call by validating ‘Access Token A’.
      1. The access token is trustworthy [6]
      2. The audience claim (aud) is ‘http://TokenExchangeApiGateway’
3. API Gateway – Client Credential Grant to Auth0
   1. API Gateway needs to get ‘Access Token B’ to authorize the call to API
      1. API requires a token with audience of ‘http://TokenExchangeApi’ so ‘Access Token A’ would be rejected
   2. Use ‘Client Credential Grant’ to request ‘Access Token B’ from Auth0
      1. The API Gateway provides it’s client id and client secret (aka service account) and includes ‘Access Token A’ as the subject token to Auth0. The subject token is what elevates this to a token exchange.
      2. The custom logic in the Auth0 hook validates the subject token [6] and if valid transfers the subject (sub) and ‘permissions’ claims to ‘http://exchange’ claim in the new token (Access Token B)
         1. Transferring claims from one token to another within the secure token server is the key concept of Token Exchange.
4. API Gateway calls API endpoint authorized by ‘Access Token B’
   1. API validates ‘Access Token B’ [6]
   2. API can trust the ‘http://exchange/’ claims because it must have been authorized by Auth0.
   3. Success! API knows the user and their roles!!

**Conclusion**

The three goals are achieved:

* The application cannot skip the API Gateway and authorize directly to the API.
  + The API will reject the access token (Access Token A) used to authenticate to ‘API Gateway’ because the audience (aud) is not correct.
* The API behind the API Gateway knows the authorized user’s context including their roles.
  + Access Token B contains the ‘http://exchange/’ claims with the user’s context.
  + The API can trust that the access token came from the trusted source (Auth0) and was not modified in transit (token signature).
* The API Gateway cannot arbitrarily act as an authorized user
  + API Gateway cannot create a valid token.
  + Auth0 will not create an access token for API Gateway with the ‘http://exchange/’ claims unless API Gateway provides a subject token with those claims.

Thus, the modular specifications of OAuth2 can be combined to provide a high level of security for the enterprise.

The last goal is especially important to me. No IT group wants the scenario where a million-dollar disbursement is fraudulently approved. There’s a user in the log but that’s based on an incoming parameter. But, who *really* made the API call? How many individuals have access to the production service account credentials?? These OAuth2 and Open-Id Connect specifications including token exchange are very valuable. There is protection from internal attacks. The developers who wrote the API and the engineers that have access to the production service accounts cannot act as another user. All those in IT can confidently say that yes, Mr. CEO, you did approve that disbursement.

**References**

1. “Tokens” <https://auth0.com/docs/tokens/id-tokens>
   1. “The audience (the aud claim) of the token is set to the application's identifier, which means that only this specific application should consume this token.”
2. “OAuth 2.0 Token Exchange” <https://tools.ietf.org/html/rfc8693>
3. “Identity Propagation in an API Gateway Architecture“ <https://medium.com/@robert.broeckelmann/identity-propagation-in-an-api-gateway-architecture-c0f9bbe9273b>
4. OAuth2 In Action by Justin Richer and Antonio Sanso
5. Auth0 Management API
   1. <https://auth0.com/docs/api/management/v2>
6. Validate Access Tokens
   1. <https://auth0.com/docs/tokens/access-tokens/validate-access-tokens>
7. Authorization Code Flow
   1. <https://auth0.com/docs/flows/authorization-code-flow>
8. Microsoft identity platform and OAuth 2.0 On-Behalf-Of flow
   1. https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-on-behalf-of-flow