### Why use OAuth2 Token Exchange with an Enterprise API Gateway?

### Authorized User

Have you ever seen a case where the user that approved a financial transaction is a parameter in the API call? The API endpoint is protected by a ‘service account’ but many employees have access to those credentials even for production. Anyone with those credentials can call the API and define an arbitrary value for the user parameter. Thus, tracing who approved the financial transaction is not lock tight.

Confidently knowing the authorized user through the entire execution chain is possible. Using the OAuth2 specification RFC 8693 Token Exchange [2] the authorized user’s context can be propagated in each token. The authorized user cannot be substituted even with service account credentials. The trustworthiness of tracing is restored.

This article shows how Rocket Mortgage uses token exchange to secure our API Gateway and APIs and includes a script so you can do it yourself in Auth0.

We 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.

Next is a short discussion of each goal. For an in-depth discussion of why this is worth the extra work I highly recommend the article “Identity Propagation in an API Gateway Architecture” by Robert Broeckelmann [3].

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

This goal excludes passing forward the 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 [1b]. These APIs have different owners who must have their own application registrations in Auth0 and independent client approvals. Having the API Gateway and API accept the same audiences would nullify the value of our security approvals and audits among other things.

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

As discussed in the introduction a key concern 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. Robert Broeckelmann explains wonderfully why knowing the originating authenticated user at each step increases the overall security of a system [3]. 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 permissions and limit their actions by parameterized API calls. This is reversed with token exchange. A service account now has few permissions until it adopts the permissions of the authorized user context that started the execution chain. In OAuth2 the user’s context is contained in signed access tokens that expire. No service account can construct a signed access token and the service account can only act as the user for a configurable period. The authorized user is maintained in a trustworthy way through the entire execution chain.

This reduces the damage if a service account’s credential is compromised. This may be referred to as a “blast radius”. If a service account can arbitrarily 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 improves protection from 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 each with their own flavor. This article uses Auth0 because that it has a great free tier option and a robust management API [5]. Other platforms also support passing the user context using token exchange [8].

A unique feature of Auth0 used in this example is the hook. The hook is custom JavaScript that executes within Auth0 during a client credential flow token request to accomplish token exchange. This allows for a high degree of customization. This is how Auth0 supports token exchange but it is different in other platforms [8].

**PowerShell Script - Auth0 Management API**

There’s nothing like seeing it work! The correlating PowerShell script ‘TokenExchangeAuth0.ps1’ 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 highly recommend setting up this example in a free Auth0 account. What you learn in Auth0 translates well 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 are not covered [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 to 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 requests a token to call the API from Auth0 using Client Credential Grant
   1. API Gateway needs to get ‘Access Token B’ to authorize the call to API
      1. The API requires a token with the audience ‘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.
      2. The custom logic in the Auth0 hook is triggered by the subject token
         1. The token is validated [6] and if valid the subject (sub) and ‘permissions’ claims are transferred to the ‘http://exchange’ claim in the new token (Access Token B)
         2. Transferring claims from one token to another *within* the secure token server is the meat of Token Exchange.
4. API Gateway calls API endpoint
   1. Include ‘Access Token B’ in the Authorization header
   2. API validates ‘Access Token B’ [6]
   3. After validating the token the API can trust the information in the ‘http://exchange/’ claims because it must have been created by Auth0.

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.
  + Within Auth0 the Application does not have approval to the API so Auth0 will reject a request from the Application for a token to access 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
  + The API Gateway cannot create a valid signed access token.
  + The API Gateway cannot get a valid signed token from Auth0 with the ‘http://exchange/’ claims *unless* API Gateway receives a token then provides it as the subject token to Auth0.

Thus, the modular specifications of OAuth2 can be combined to provide a high level of security for the enterprise. All protected operations and tracing are based on the token which is trustworthy. There is protection from internal attacks.

### References

1. Tokens
   1. <https://auth0.com/docs/tokens/id-tokens>
   2. 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
   1. <https://tools.ietf.org/html/rfc8693>
3. Identity Propagation in an API Gateway Architecture
   1. <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. Other Platforms:
   1. 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>
   2. Ping Federate
      1. <https://docs.pingidentity.com/bundle/pingfederate-100/page/ohu1576176732815.html>