Check out the free virtual workshops on how to take your SaaS app to the next level in the enterprise-ready identity journey!

oauth what-is-oauth

What is the OAuth 2.0 Authorization Code Grant Type?

Aaron Parecki

April 10, 2018

4 MIN READ

The Authorization Code Grant Type is probably the most common of the OAuth 2.0 grant types that you'll encounter. It is used by both web apps and native apps to get an access token after a user authorizes an app.

This post is the first part of a series where we explore frequently used OAuth 2.0 grant types. If you want to back up a bit and learn more about OAuth 2.0 before we dive in, check out What the Heck is OAuth?, also on the Okta developer blog.

What is an OAuth 2.0 Grant Type?

In OAuth 2.0, the term "grant type" refers to the way an application gets an access token. OAuth 2.0 defines several grant types, including the authorization code flow. OAuth 2.0 extensions can also define new grant types.

The Authorization Code Flow

The Authorization Code grant type is used by web and mobile apps. It differs from most of the other grant types by first requiring the app launch a browser to begin the flow. At a high level, the flow has the following steps:

- The application opens a browser to send the user to the OAuth server
- The user sees the authorization prompt and approves the app's request
- The user is redirected back to the application with an authorization code in the query string
- The application exchanges the authorization code for an access token

Get the User's Permission

OAuth is all about enabling users to grant limited access to applications. The application first needs to decide which permissions it is requesting, then send the user to a browser to get their permission. To begin the authorization flow, the application constructs a URL like the following and opens a browser to that URL.

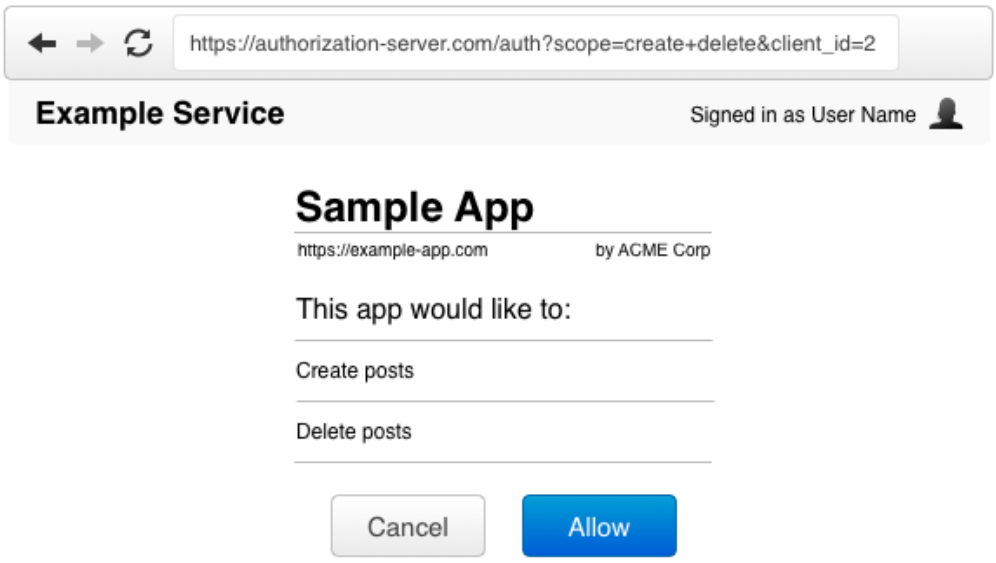
```
https://authorization-server.com/auth
 ?response_type=code
 &client_id=29352915982374239857
 &redirect_uri=https%3A%2F%2Fexample-app.com%2Fcallback
 &scope=create+delete
 &state=xcoiv98y2kd22vusuye3kch
```

Here's each query parameter explained:

- `response_type=code` - This tells the authorization server that the application is initiating the authorization code flow.
- `client_id` - The public identifier for the application, obtained when the developer first registered the application.

- `scope` - One or more space-separated strings indicating which permissions the application is requesting. The specific OAuth API you're using will define the scopes that it supports.
- `state` - The application generates a random string and includes it in the request. It should then check that the same value is returned after the user authorizes the app. This is used to prevent CSRF attacks.

When the user visits this URL, the authorization server will present them with a prompt asking if they would like to authorize this application's request.



Redirect Back to the Application

If the user approves the request, the authorization server will redirect the browser back to the `redirect_uri` specified by the application, adding a `code` and `state` to the query string.

For example, the user will be redirected back to a URL such as

```
https://example-app.com/redirect
 ?code=g0ZGZmNjVmOWIjNTk2NTk4ZTYyZGI3
 &state=xcoiv98y2kd22vusuye3kch
```

This protects against CSRF and other related attacks.

The `code` is the authorization code generated by the authorization server. This code is relatively short-lived, typically lasting between 1 to 10 minutes depending on the OAuth service.

Exchange the Authorization Code for an Access Token

We're about ready to wrap up the flow. Now that the application has the authorization code, it can use that to get an access token.

The application makes a POST request to the service's token endpoint with the following parameters:

- `grant_type=authorization_code` - This tells the token endpoint that the application is using the Authorization Code grant type.
- `code` - The application includes the authorization code it was given in the redirect.
- `redirect_uri` - The same redirect URI that was used when requesting the code. Some APIs don't require this parameter, so you'll need to double check the documentation of the particular API you're accessing.
- `client_id` - The application's client ID.
- `client_secret` - The application's client secret. This ensures that the request to get the access token is made only from the application, and not from a potential attacker that may have intercepted the authorization code.

The token endpoint will verify all the parameters in the request, ensuring the code hasn't expired and that the client ID and secret match. If everything checks out, it will generate an access token and return it in the response!

```
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"MTQ0NjJkZmQ5OTM2NDE1ZTZjNGZmZjI3",
  "token_type":"bearer",
  "expires_in":3600,
  "refresh_token":"IwOGYzYTlmM2YxOTQ5MGE3YmNmMDFkNTVk",
  "scope":"create delete"
}
```

The Authorization Code flow is complete! The application now has an access token it can use when making API requests.

When to use the Authorization Code Flow

The Authorization Code flow is best used in web and mobile apps. Since the Authorization Code grant has the extra step of exchanging the authorization code for the access token, it provides an additional layer of security not present in the Implicit grant type.

If you're using the Authorization Code flow in a mobile app, or any other type of application that can't store a client secret, then you should also use the PKCE extension, which provides protections against other attacks where the authorization code may be intercepted.

The code exchange step ensures that an attacker isn't able to intercept the access token, since the access token is always sent via a secure backchannel between the application and the OAuth server.

Learn More About OAuth and Okta

You can learn more about OAuth 2.0 on OAuth.com, or check out any of these resources to get started building!

- Secure your SPA with Spring Boot and OAuth

Or hit up Okta's OIDC/OAuth 2.0 API for specific information on how we support OAuth. And as always, follow us on Twitter @oktadev for more great content.

PS: We recently built a new security site where we're publishing lots of other security-focused articles (like this one). You should cehck it out!

Aaron Parecki

Aaron Parecki is a Senior Security Architect at Okta. He is the author of OAuth 2.0 Simplified, and maintains oauth.net. He regularly writes and gives talks about OAuth and online security. He is an editor of several internet specs, and is the co-founder of IndieWebCamp, a conference focusing on data ownership and online identity. Aaron has spoken at conferences around the world about OAuth, data ownership, quantified self, and home automation, and his work has been featured in Wired, Fast Company and more.

Previous post

Next post

Okta Developer Blog Comment Policy

We welcome relevant and respectful comments. Off-topic comments may be removed.