OAuth 2.0

OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. This specification and its extensions are being developed within the IETF OAuth Working Group.

OAuth 2.1 is an in-progress effort to consolidate OAuth 2.0 and many common extensions under a new name.

Questions, suggestions and protocol changes should be discussed on the mailing list.

OAuth Grant Types

The OAuth framework specifies several grant types for different use cases, as well as a framework for creating new grant types.

The most common OAuth grant types are listed below.

- Authorization Code
- PKCE
- Client Credentials
- Device Code
- Refresh Token

More resources

- The Nuts and Bolts of OAuth (Video Course) - Aaron Parecki
- Grant Types (aaronparecki.com)
- A Guide to OAuth 2.0 Grants (alexbilbie.com)

Legacy

- Implicit Flow
- Password Grant

OAuth 2.0 Authorization Code Grant

tools.ietf.org/html/rfc6749#section-1.3.1

The Authorization Code grant type is used by confidential and public clients to exchange an authorization code for an access token.

After the user returns to the client via the redirect URL, the application will get the authorization code from the URL and use it to request an access token.

It is recommended that all clients use the PKCE extension with this flow as well to provide better security.

OAuth 2.0 Client Credentials Grant

tools.ietf.org/html/rfc6749#section-4.4

The Client Credentials grant type is used by clients to obtain an access token outside of the context of a user.
This is typically used by clients to access resources about themselves rather than to access a user's resources.

OAuth Access Tokens

datatracker.ietf.org/doc/html/rfc6749#section-1.4

An OAuth Access Token is a string that the OAuth client uses to make requests to the resource server.

Access tokens do not have to be in any particular format, and in practice, various OAuth servers have chosen many different formats for their access tokens.

Access tokens may be either "bearer tokens" or "sender-constrained" tokens. Sender-constrained tokens require the OAuth client to prove possession of a private key in some way in order to use the access token, such that the access token by itself would not be usable.

There are a number of properties of access tokens that are fundamental to the security model of OAuth:

- Access tokens must not be read or interpreted by the OAuth client. The OAuth client is not the intended audience of the token.
- Access tokens do not convey user identity or any other information about the user to the OAuth client.
- Access tokens should *only* be used to make requests to the resource server. Additionally, ID tokens must *not* be used to make requests to the resource server.

OAuth Refresh Tokens

datatracker.ietf.org/doc/html/rfc6749#section-1.5

An OAuth Refresh Token is a string that the OAuth client can use to get a new access token without the user's interaction.

A refresh token must not allow the client to gain any access beyond the scope of the original grant. The refresh token exists to enable authorization servers to use short lifetimes for access tokens without needing to involve the user when the token expires.

OAuth Scopes

tools.ietf.org/html/rfc6749#section-3.3

Scope is a mechanism in OAuth 2.0 to limit an application's access to a user's account. An application can request one or more scopes, this information is then presented to the user in the consent screen, and the access token issued to the application will be limited to the scopes granted.

The OAuth spec allows the authorization server or user to modify the scopes granted to the application compared to what is requested, although there are not many examples of services doing this in practice.

OAuth does not define any particular values for scopes, since it is highly dependent on the service's internal architecture and needs.

What is OAuth 2.0?

OAuth 2.0, which stands for “Open Authorization”, is a standard designed to allow a website or application to access resources hosted by other web apps on behalf of a user. It replaced OAuth 1.0 in 2012 and is now the de facto industry standard for online authorization. OAuth 2.0 provides consented access and restricts actions of what the client app can perform on resources on behalf of the user, without ever sharing the user's credentials.

Although the web is the main platform for OAuth 2, the specification also describes how to handle this kind of delegated access to other client types (browser-based applications, server-side web applications, native/mobile apps, connected devices, etc.)

Principles of OAuth2.0

OAuth 2.0 is an authorization protocol and NOT an authentication protocol. As such, it is designed primarily as a means of granting access to a set of resources, for example, remote APIs or user's data.

OAuth 2.0 uses Access Tokens. An **Access Token** is a piece of data that represents the authorization to access resources on behalf of the end-user. OAuth 2.0 doesn't define a specific format for Access Tokens. However, in some contexts, the JSON Web Token (JWT) format is often used. This enables token issuers to include data in the token itself. Also, for security reasons, Access Tokens may have an expiration date.

OAuth2.0 Roles

The idea of roles is part of the core specification of the OAuth2.0 authorization framework. These define the essential components of an OAuth 2.0 system, and are as follows:

- **Resource Owner**: The user or system that owns the protected resources and can grant access to them.
- **Client**: The client is the system that requires access to the protected resources. To access resources, the Client must hold the appropriate Access Token.
- **Authorization Server**: This server receives requests from the Client for Access Tokens and issues them upon successful authentication and consent by the Resource Owner. The authorization server exposes two endpoints: the Authorization endpoint, which handles the interactive authentication and consent of the user, and the Token endpoint, which is involved in a machine to machine interaction.
- **Resource Server**: A server that protects the user's resources and receives access requests from the Client. It accepts and validates an Access Token from the Client and returns the appropriate resources to it.

OAuth 2.0 Scopes

Scopes are an important concept in OAuth 2.0. They are used to specify exactly the reason for which access to resources may be granted. Acceptable scope values, and which resources they relate to, are dependent on the Resource Server.

OAuth 2.0 Access Tokens and Authorization Code

The OAuth 2 Authorization server may not directly return an Access Token after the Resource Owner has authorized access. Instead, and for better security, an **Authorization Code** may be returned, which is then exchanged for an Access Token. In addition, the Authorization server may also issue a Refresh Token with the Access Token. Unlike Access Tokens, Refresh Tokens normally have long expiry times and may be exchanged for new Access Tokens when the latter expires. Because Refresh Tokens have these properties, they have to be stored securely by clients.

How Does OAuth 2.0 Work?

At the most basic level, before OAuth 2.0 can be used, the Client must acquire its own credentials, a *client id* and *client secret*, from the Authorization Server in order to identify and authenticate itself when requesting an Access Token.

Using OAuth 2.0, access requests are initiated by the Client, e.g., a mobile app, website, smart TV app, desktop application, etc. The token request, exchange, and response follow this general flow:

1. The Client requests authorization (*authorization request*) from the Authorization server, supplying the client id and secret to as identification; it also provides the scopes and an endpoint URI (*redirect URI*) to send the Access Token or the Authorization Code to.

2. The Authorization server authenticates the Client and verifies that the requested scopes are permitted.
3. The Resource owner interacts with the Authorization server to grant access.
4. The Authorization server redirects back to the Client with either an Authorization Code or Access Token, depending on the grant type, as it will be explained in the next section. A Refresh Token may also be returned.
5. With the Access Token, the Client requests access to the resource from the Resource server.

Grant Types in OAuth 2.0

In OAuth 2.0, **grants** are the set of steps a Client has to perform to get resource access authorization. The authorization framework provides several grant types to address different scenarios:

- **Authorization Code grant**: The Authorization server returns a single-use **Authorization Code** to the Client, which is then exchanged for an Access Token. This is the best option for traditional web apps where the exchange can securely happen on the server side. The Authorization Code flow might be used by Single Page Apps (SPA) and mobile/native apps. However, here, the client secret cannot be stored securely, and so authentication, during the exchange, is limited to the use of *client id* alone. A better alternative is the *Authorization Code with PKCE* grant, below.
- **Implicit Grant**: A simplified flow where the Access Token is returned directly to the Client. In the Implicit flow, the authorization server may return the Access Token as a parameter in the callback URI or as a response to a form post. The first option is now deprecated due to potential token leakage.
- **Authorization Code Grant with Proof Key for Code Exchange (PKCE)**: This authorization flow is similar to the *Authorization Code* grant, but with additional steps that make it more secure for mobile/native apps and SPAs.
- **Resource Owner Credentials Grant Type**: This grant requires the Client first to acquire the resource owner's credentials, which are passed to the Authorization server. It is, therefore, limited to Clients that are completely trusted. It has the advantage that no redirect to the Authorization server is involved, so it is applicable in the use cases where a redirect is infeasible.
- **Client Credentials Grant Type**: Used for non-interactive applications e.g., automated processes, microservices, etc. In this case, the application is authenticated per se by using its client id and secret.
- **Device Authorization Flow**: A grant that enables use by apps on input-constrained devices, such as smart TVs.
- **Refresh Token Grant**: The flow that involves the exchange of a Refresh Token for a new Access Token.