OAuth 2.0 Authorization Framework

In this article

The OAuth 2.0 authorization framework is a protocol that allows a user to grant a third-party web site or application access to the user's protected resources, without necessarily revealing their long-term credentials or even their identity.

OAuth introduces an authorization layer and separates the role of the client from that of the resource owner. In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server and is issued a different set of credentials than those of the resource owner. Instead of using the resource owner's credentials to access protected resources, the client obtains an access token--a string denoting a specific scope, lifetime, and other access attributes. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner. Then the client uses the access token to access the protected resources hosted by the resource server.

Auth0 generates access tokens for API authorization scenarios, in JSON web token (JWT) format. The permissions represented by the access token, in OAuth terms, are known as scopes. When an application authenticates with Auth0, it specifies the scopes it wants. If those scopes are authorized by the user, then the access token will represent these authorized scopes.

Roles

An OAuth 2.0 flow has the following roles:

- **Resource Owner**: Entity that can grant access to a protected resource. Typically, this is the end-user.
- **Resource Server**: Server hosting the protected resources. This is the API you want to access.
- **Client**: Application requesting access to a protected resource on behalf of the Resource Owner.
- **Authorization Server**: Server that authenticates the Resource Owner and issues access tokens after getting proper authorization. In this case, Auth0.

Grant types

OAuth 2.0 defines four flows to get an access token. These flows are called grant types. Deciding which one is suited for your case depends mostly on your application type.

- Authorization Code Flow: used by Web Apps executing on a server. This is also used by mobile apps, using the Proof Key for Code Exchange (PKCE) technique.
- Implicit Flow with Form Post: used by JavaScript-centric apps (Single-Page Applications) executing on the user's browser.

- Resource Owner Password Flow: used by highly-trusted apps.
- Client Credentials Flow: used for machine-to-machine communication.

The specification also provides an extensibility mechanism for defining additional grant types. To learn more about how each grant type works and when it should be used, see Authentication and Authorization Flows.

Endpoints

OAuth 2.0 uses two endpoints: the `/authorize` endpoint and the `/oauth/token` endpoint.

Authorization endpoint

The `/authorize` endpoint is used to interact with the resource owner and get the authorization to access the protected resource. To better understand this, imagine that you want to log in to a service using your Google account. First, the service redirects you to Google in order to authenticate (if you are not already logged in) and then you will get a consent screen, where you will be asked to authorize the service to access some of your data (protected resources); for example, your email address and your list of contacts.

The request parameters of the `/authorize` endpoint are:

Parameter	Description
`response_type`	Tells the authorization server which grant to execute.

Parameter	Description
`response_mode`	(Optional) How the result of the authorization request is formatted. Values: - `query` : for Authorization Code grant. `302 Found` triggers redirect. - `fragment` : for Implicit grant. `302 Found` triggers redirect. - `form_post` : `200 OK` with response parameters embedded in an HTML form as hidden parameters. - `web_message` : For Silent Authentication. Uses HTML5 web messaging.
`client_id`	The ID of the application that asks for authorization.
`redirect_uri`	Holds a URL. A successful response from this endpoint results in a redirect to this URL.
`scope`	A space-delimited list of permissions that the application requires.
`state`	An opaque value, used for security purposes. If this request parameter is set in the request, then it is returned to the application as part of the `redirect_uri` .

This endpoint is used by the Authorization Code and the Implicit grant types. The authorization server needs to know which grant type the application wants to use since it affects the kind of credential it will issue:

- For the Authorization Code grant, it will issue an authorization code (which can later be exchanged with an access token).
- For the Implicit grant, it will issue an access token.

An access token is an opaque string (or a JWT in an Auth0 implementation) that denotes who has authorized which permissions (scopes) to which application. It is meant to be exchanged with an access token at the `/oauth/token` endpoint. To inform the authorization server which grant type to use, the `response_type` request parameter is used as follows:

- For the Authorization Code grant, use `response_type=code` to include the authorization code.
- For the Implicit grant, use `response_type=token` to include an access token. An alternative is to use `response_type=id_token token` to include both an access token and an ID token.

An ID token is a JWT that contains information about the logged in user. It was introduced by OpenID Connect (OIDC).

The OAuth 2.0 Multiple Response Type Encoding Practices specification added a parameter that specifies how the result of the authorization request is formatted. This parameter is called `response_mode`. It is optional and can take the following values:

Value	Description
`query`	This is the default for Authorization Code grant. A successful response is `302 Found` which triggers a redirect to the `redirect_uri`. The response parameters are embedded in the query component (the part after `?`) of the `redirect_uri` in the `Location` header. For example: `HTTP/1.1 302 Found` `Location: https://my-redirect-uri.callback?code=js89p2x1` where the authorization ocde is `js89p21`.
`fragment`	This is the default for Implicit grant. A successful response is `302 Found`, which triggers a redirect to the `redirect_uri` (which is a request parameter). The response parameters are embedded in the fragment component (the part after `#`) of the `redirect_uri` in the `Location` header. For example: `HTTP/1.1 302 Found` `Location: https://my-redirect-uri/callback#access_token=eyB...78f&token_type=Bearer&expires_in=3600`.

Value	Description
`form_post`	The response mode is defined by the OAuth 2.0 Form Post Response Mode specification. A successful response is `200 OK` and the parameters are embedded in an HTML form as hidden params. The `action` of the form is the `redirect_uri` and the `onload` attribute is configured to submit the form. After the HTML is loaded by the browser, a redirect to the `redirect_uri` is done.
`web_message`	This response mode is defined in OAuth 2.0 Web Message Response Mode specification. It uses HTML5 Web Messaging instead of the redirect for the authorization response from the /authorization endpoint. This is particularly useful when using Silent Authentication. To do this response mode, you must register your app's URL at the **Allowed Web Origins** field in your Auth0 application settings.

Token endpoint

The `/oauth/token` endpoint is used by the application in order to get an access token or a refresh token. It is used by all flows except for the Implicit Flow because in that case an access token is issued directly.

- In the Authorization Code Flow, the application exchanges the authorization code it got from the authorization endpoint for an access token.
- In the Client Credentials Flow and Resource Owner Password Credentials Grant Exchange, the application authenticates using a set of credentials and then gets an access token.

State parameters

Authorization protocols provide a `state` parameter that allows you to restore the previous state of your application. The `state` parameter preserves some state object set by the client in the Authorization request and makes it available to the client in the response. The primary reason for using the state parameter is to mitigate CSRF attacks. See Use OAuth 2.0 State Parameters for details.

Keep reading

- Prevent Attacks and Redirect Users with OAuth 2.0 State Parameters
- Which OAuth 2.0 Flow Should I Use?

CONTACT

10800 NE 8th Street

Suite 600

Bellevue, WA 98004

+1 (888) 235-2699

+1 (425) 312-6521

+44 (0) 33-3234-1966