Auth0 Docs

OAuth 2.0 Authorization Framework

In this article

OAuth 2.0 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. The client uses the access token to access the protected resources hosted by the resource server.

Access Token Format

By default, Auth0 generates Access Tokens for API Authorization scenarios in JSON Web Token (JWT) format. JWTs contain three parts: a header, a payload, and a signature:

- The header contains metadata about the type of token and the cryptographic algorithms used to secure its contents.
- The payload contains a set of claims, which are statements about the permissions that should be allowed, and other information like the intended audience and the expiration time.

- The sig en is trustworthy and has not been tampered with.

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. For example, a Contacts API may accept three different authorization levels specifying the corresponding scopes:

- Reading contacts (`read:contacts`)
- Creating contacts (scope `create:contacts`)
- Deleting contacts (scope `delete:contacts`)

For more information refer to Scopes.

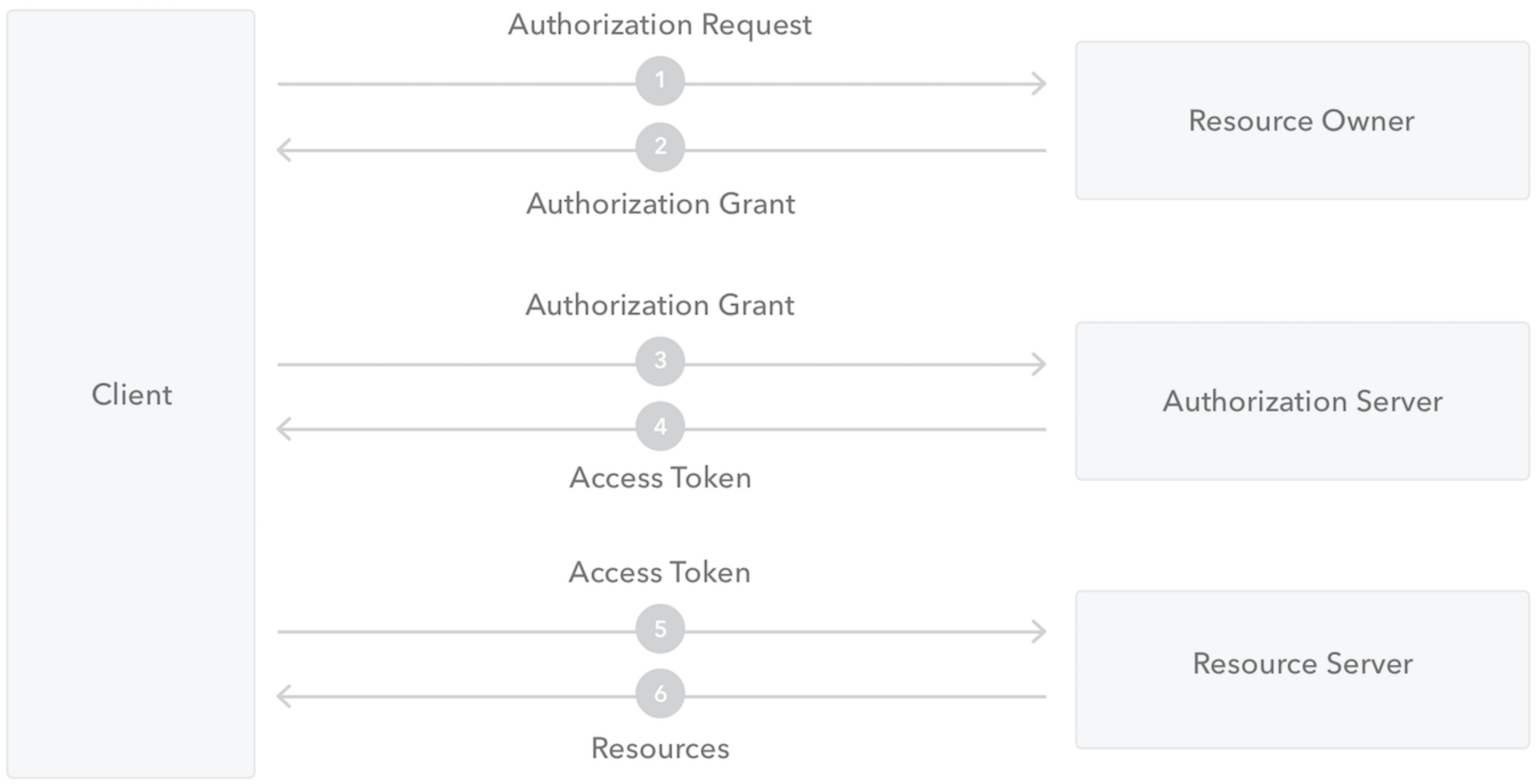
OAuth roles

In any OAuth 2.0 flow we can identify the following roles:

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

Protocol flow

We will now have a more detailed look on how the protocol works. As we will see in a while, OAuth has many "flavors" (called authorization grant types) that you can use. For now we will have a more generic look into the flow.



1. The Application (Client) asks for authorization from the Resource Owner in order to access the resources.

2. Provided tl is access, the Application receives an **Authorization Grant**. This is a credential representi n.

3. The Application requests an **Access Token** by authenticating with the Authorization Server and giving the Authorization Grant.

4. Provided that the Application is successfully authenticated and the Authorization Grant is valid, the Authorization Server issues an Access Token and sends it to the Application.

5. The Application requests access to the protected resource by the Resource Server, and authenticates by presenting the Access Token.

6. Provided that the Access Token is valid, the Resource Server serves the Application's request.

Authorization grant types

The OAuth 2.0 Authorization Framework specification 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 the type of your application.

- Authorization Code: 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: used by JavaScript-centric apps (Single-Page Applications) executing on the user's browser.

- Resource Owner Password Credentials: used by trusted apps.

- Client Credentials: used for machine-to-machine communication.

The specifica chanism for defining additional types.

For details on how each grant type works and when it should be used refer to API Authorization.

OAuth endpoints

OAuth 2.0 utilizes two endpoints: the **Authorization** endpoint and the **Token** endpoint.

Authorization endpoint

The Authorization 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 Authorization endpoint are:

- `response_type` : Tells the authorization server which grant to execute. Refer to the How Response Type Works paragraph for details.
- `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` .

How respo

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

To inform the authorization server which grant type to use, the `response_type` request parameter is used as follows:

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

ID Token

The ID Token is a JWT that contains information about the logged in user. It was introduced by **OpenID Connect (OIDC)**. For more info, see OpenID Connect and ID Token.

How response mode works

The OAuth 2. Practices specification added a new parameter that specifies how the result of the authorizatior is called `response_mode`, it's optional and can take the following values:

- `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, a response might be:

  ```
  HTTP/1.1 302 Found
  Location: https://my-redirect-uri/callback?code=js89p2x1
  ```

  Was this helpful? Yes / No

  Where, the `redirect_uri` is `https://my-redirect-uri/callback` and the authorization code is `js89p2x1`.

- `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, a response might be:

  ```
  HTTP/1.1 302 Found
  Location: https://my-redirect-uri/callback#access_token=eyB...78f&token_type=Bearer&expires_in=3600
  ```

  Was this helpful? Yes / No

Where, the `irect-uri/callback`, the Access Token is `eyB...78f`, it's a Bearer token and it expires in

- `form_post`: This response mode is defined in the OAuth 2.0 Form Post Response Mode specification. A successful response is `200 OK` and the response 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.

  For example, a response might be:

```
HTTP/1.1 200 OK
<html>
 <head><title>Submit</title></head>
 <body onload="javascript:document.forms[0].submit()">
  <form method="post" action="https://my-redirect-uri.com/callback">
    <input type="hidden" name="state" value="klsdfY78FVN3sl6DWSjsdhfsd8r67832nb"/>
    <input type="hidden" name="id_token" value="eyJ...plD"/>
  </form>
 </body>
</html>
```

Was this helpful? Yes / No

- `web_message`: This response mode is defined in the 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 Silen e mode you have to register your app's URL at the **Allowed Web Origins** field of your Applicatio

Token endpoint

The 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 (since 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, the application authenticates using a set of credentials and then gets an access token.

Keep reading

- Which OAuth 2.0 flow should I use?
- API Authorization
- State Parameter
- Mitigate CSRF Attacks
- Redirect Users

Was this article helpful?

✓ YES ✗ NO

Any suggestion or typo? Edit on GitHub

Press

LEARN

Availability & Trust

Security

White Hat

API Explorer

MORE

Help & Support

Professional Services

Documentation

Open Source

WordPress

CONTACT

10800 NE 8th Street

Suite 600

+1 (888) 235-2699

+1 (425) 312-6521

Bellevue, WA 98

+44 (0) 33-3234-1966

Follow 14 086 Follow 5 412 Like 14 395

Privacy Policy Terms of Service