Protecting APIs with OAuth 2.0

Protecting APIs with OAuth 2.0

On this page:

- Background
- Registering a scope
- Parsing and validating an access token
- Performing token introspection/validation
- Example Ecosystem
  - Components
  - Procedure
- Related Articles

Background

The MIT OIDC server https://oidc.mit.edu/ can serve as an external authorization server for APIs. Direct any clients who wish to have access your protected API to the MIT OIDC server to get OAuth 2.0 access tokens.

This assumes you have successfully registered a client with the OIDC server.

Registering a scope

In addition to the scopes needed to obtain end-user information, the server supports additional scopes. Your API should register a scope or a structured scope with the server.

Parsing and validating an access token

All access tokens issued by the MIT OIDC server are asymmetrically signed JWTs. If a protected resource simply wants to verify the origin of a given token, it can parse the JWT and validate the signature using JWS and the server's published public key. This process is described in [istcontrib:Logging in users to your application using OpenID Connect].

- If validation fails, your API should deny access.

Performing token introspection/validation

All protected resources that make use of the token introspection endpoint must be registered with server. The MIT OIDC server supports registration through a user-facing developer portal at https://oidc.mit.edu/manage/dev/dynreg. Be sure to add any scopes required for accessing the API to the registration request. For example, if your client requests tokens with the scope "my_api", then the protected resource must also be registered with that scope, otherwise it will be forbidden from validating those tokens.

Successful registration will produce a **client_id** and **client_secret** which are used in the call to the introspection endpoint.

To valuate a token using token introspection, make an HTTP POST to **https://oidc.mit.edu/introspect**. Include the **client_id** and **client_secret** received during registration as the username and password of an HTTP Basic Authentication header. Include the token being checked as an HTTP Form Parameter with the name **token** in the body of the request. The server will respond with a JSON object containing the following fields:

- **active**: whether or not the token is currently active
  - If the token is not active, your API should immediately deny access.
- **sub**: the 'subject' of the user who authorized this token
- **user_id**: the username of the user who authorized this token (specific to MIT's deployment)
- **client_id**: the identifier of the client that requested this token
- **scope**: a space-separated list of scopes that the token was issued with

Example Ecosystem

This example may help clarify things. Imagine you are writing an API, and wish to protect it with OAuth 2.0, and wish to allow MIT community members to use it.

Components

- `https://example.mit.edu/developers` – A web page where community members can register to use your API, and receive access tokens. It may also contain API documentation.
    - This is an OpenID Connect **client**.
- `https://example.mit.edu/api/` – The root of your API. API consumers make requests to endpoints such as `https://example.mit.edu/api/things` and `https://example.mit.edu/api/stuff`
    - This is an OpenID Connect **protected resource**.

Procedure

1. Register a client with the OpenID Connect server. Within the client registration, add scopes that your API will use. You must register at least one scope. For the purposes of this example, assume the scope is "test". You should add the other standard scopes only if your client will be authenticating end users and requiring user information.
2. Register a protected resource with the OpenID Connect server. It should be registered with the same scopes as your client.
3. When someone visits `https://example.mit.edu/developers`, create a link that directs them through the authorization flow, as described in "[istcontrib:Logging in users to your application using OpenID Connect]". When requesting scopes in the authorization scope, be sure to add the scopes for your API. (e.g. "test")
4. Provide the returned access_token to the user.
5. The user then goes on to access `https://example.mit.edu/api/things` and presents the access_token.
6. The API, which has been registered as a protected resource, and assigned its own client_id and client_secret, takes the access_token and presents it to the introspection endpoint.
7. Using the information contained in the response from the endpoint, the API decides whether or not to permit access.

Related Articles

- [MIT's OpenID Connect Server Information]
- [istcontrib:Logging in users to your application using OpenID Connect]