Accessing Webfleet OAuth APIs

Reference Guide

1587719116942 - 1.2 - 2023-07-06

Solutions for your journey

Contents

Getting started

Introduction

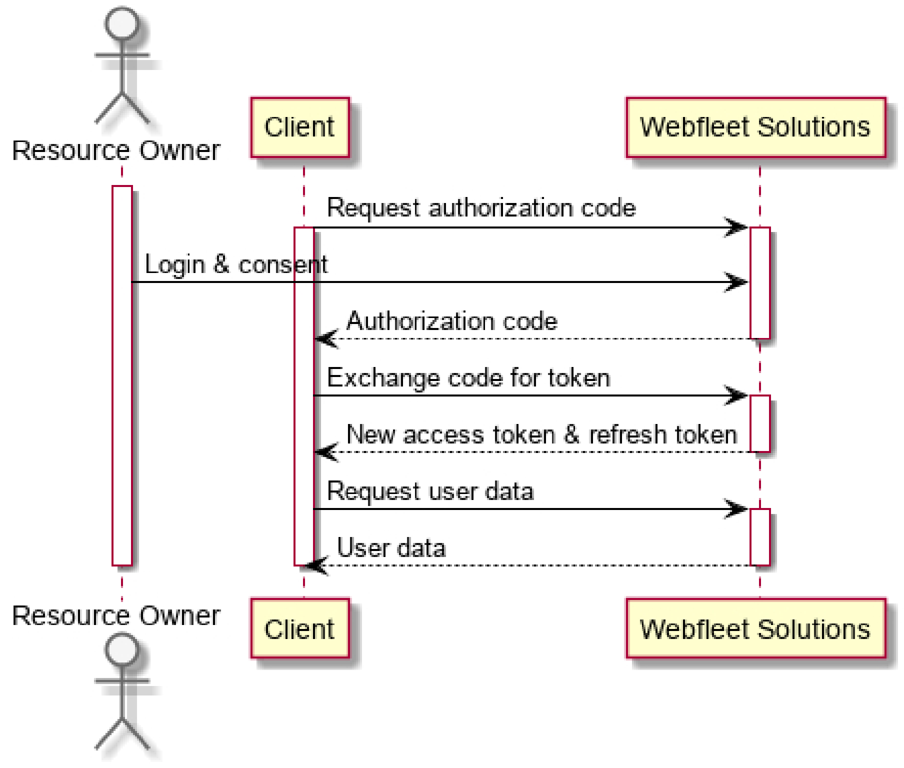
Webfleet APIs are secured using OAuth2 and partially inspired by OpenId Connect Core 1.0. Samples showing how to integrate can be found in the official Webfleet Github account.

Before starting to integrate with Webfleet APIs, a pair of OAuth 2.0 client credentials are required. These credentials identify your application as a client (OAuth 2.0. Ask the Webfleet customer support about the right procedure to register your application as a client to obtain your credentials.

Client credentials cannot be used to log in to Webfleet. These identify a client application and do not refer to a customer's credentials.

Once provided with a pair of client credentials you can request through **Webfleet Authorization Server** authorisation and token endpoints a claim granting you access to Webfleet APIs.

Nevertheless, customers will need to grant access to user data following OAuth 2.0 Authorization code grant flow. The diagram below shows Authorization code flow steps.



At completion of this flow an access token and refresh tokens are granted to the client providing access to customer's data he gave consent for on his behalf. Use Bearer token (RFC 6750) in an authorisation header (RFC 6750 section 2.1) to access Webfleet APIs.

Example using Bearer token authorization header (RFC 6750 section 2.1):

```
GET /api HTTP/1.1
Host: api.webfleet.com
Authorization: Bearer eyJhbGciOiJI...
```

Terminology

Access token

A token used to access protected resources.

Authorization code

An intermediary token generated when a user authorises a client to access protected resources on their behalf. The client receives this token and exchanges it for an access token.

Authorization server

A server which issues access tokens after successfully authenticating a client and resource owner, and authorising the request.

Client

An application which accesses protected resources on behalf of the resource owner (such as a user). The client could be hosted on a server, desktop, mobile or other device. Also known as Relying Party in OpenId Connect Core 1.0.

Grant

A Grant is a method of acquiring an access token.

JWT

A JSON Web Token is a method for representing claims securely between two parties as defined in RFC 7519.

Offline session

It is like a standard session but is controlled by an offline token. It can be maintained indefinitely if the offline token has not expired and will keep performing actions even if the user is not online.

Offline token

A special kind of refresh token that allows the application to grant new access tokens even after the user is logged out and the active session is expired. This action is useful if your application needs to perform offline actions on behalf of the user such as data backups.

Refresh token

A token requested as part of the process of obtaining an access token. When the access token expires the application can request a new one using this refresh token and the client id, so the user retains access to the resources without granting permissions again.

Resource owner

The user (typically a Webfleet customer) who authorises an application to access their account. The access of the application to the account of the user is limited to the scope of the authorisation granted (e.g. read or write access).

Resource server

A server which sits in front of protected resources (for example vehicle data, drivers) and is capable of accepting and responding to protected resource requests using access tokens.

Scope

A scope is a mechanism to limit the access of an application to the account of a user (e.g. read profile information).

Webfleet Authorization Server endpoints

Find below Webfleet Authorization server endpoints used to authenticate and revoke access in Webfleet.

Authorization endpoint	https://login.webfleet.com/auth/realms/webfleet/protocol/openid-connect/auth
Token endpoint	https://login.webfleet.com/auth/realms/webfleet/protocol/openid-connect/token
Token revocation endpoint	https://login.webfleet.com/auth/realms/webfleet/protocol/openid-connect/revoke
JWK keys endpoint	https://login.webfleet.com/auth/realms/webfleet/protocol/openid-connect/certs

Working with Webfleet Authorization Server

Obtaining an access and refresh token

Webfleet APIs use OAuth 2.0. Clients must use the OAuth 2.0 Authorization code grant flow to obtain an access token which is a self contained signed structure following JWT specification. This flow ensures resource owners privacy and explicitly represents a mutual trust agreemet between all parties; resource owner, client and Webfleet.

Below are steps to obtain an access token, steps below follow standard specification and are documented for better understanding of the process. It is recommended to use an available OAuth library, a list of available client libraries for different languages can be found in https://oauth.net/code/.

Triggering OAuth 2.0 authorisation code flow

The authentication flow is triggered by redirecting resource owner's user agent to Webfleet Auth authorization endpoint with the following parameters:

Parameter	Description
redirect_uri	This is the callback URI which will receive the authorisation code from the user's authorisation process. This parameter must be provided during registration process.
response_type	OAuth 2.0 grant flow to use. Use `code` as parameter value to trigger OAuth 2.0 Authorization code grant flow.
client_id	OAuth client credentials username, provided during the partnership registration process.
state	A random state value that is used to correlate/validate the request in the callback later. This parameter is optional.
scope	List of OAuth 2.0 scopes. This parameter is optional. By default, clients will not require this parameter, default scopes are always assigned to the client, regardless of the optional scopes. Additional scopes might be required to access other APIs or apply new features. Consult related API documentation to learn more about its required scopes. The use of the scope “offline_access” is needed to obtain offline tokens.

Example using the authorisation endpoint to trigger OAuth 2.0 Authorization code grant flow

```
GET
http://auth/realms/webfleet/protocol/openid-connect/auth?scope=<offline_ac-
cess>&redirect_uri=<your_redirect_uri>&client_id=<your_client_id>&re-
sponse_type=code&state={random} HTTP/1.1
Host: https://login.webfleet.com
```

This request will redirect the user agent to a login page on the Webfleet servers in which resource owner credentials must be entered. This avoids entering credentials on 3rd party software keeping them from potential leaks.

Obtaining an authorisation code

At the end of this process resource owner's browser is redirected to the uri provided in `redirect_uri` parameter with the authorisation code. This authorisation code can then be used to request an access token from the token endpoint.

In the previous example the user agent will have been redirected to:

```
https://integrator.example.com/cb?code=auth_code&state=some_value
```

At this point user interaction is no longer necessary and it depends on application requirements whether the next steps need to happen in the same flow as the user's interaction or can be executed in the background.

The authorisation code received can be exchanged for an access token only once using the token endpoint. After its usage the authorisation code is invalidated and cannot be reused.

Exchanging authorization code for access and refresh tokens

To request an access token using an authorization code client credentials can provided in a Basic Authorization header, base64 encoded (RFC 7235) or as parameters in the request payload.

Authorization code and the redirect_uri parameter must be provided to enforce security, validating it's the same as registered by the OAuth client. The token exchange request is form encoded and requires the following parameters:

Parameter	**Description**
grant_type	OAuth 2.0 grant type. Use `authorization_code` as parameter value to follow OAuth 2.0 Authorization code grant flow.
code	The authorisation code received in previous step.
redirect_uri	The redirect URI associated to the OAuth client provided during the partner registration process.
client_id	OAuth client credentials username provided during registration process.
scope	List of OAuth 2.0 scopes. This parameter is optional. By default, clients will not require this parameter, default scopes are always assigned to the client, regardless of the optional scopes. Additional scopes might be required to access other APIs or apply new features. Consult related API documentation to learn more about its required scopes. The use of the scope “offline_access” is needed to obtain offline tokens.
client_secret	OAuth client credentials password provided during registration process.

Example:

```
POST /auth/realms/webfleet/protocol/openid-connect/token HTTP/1.1
Host: https://login.webfleet.com
Content-type: application/x-www-form-urlencoded
Accept: application/json

grant_type=authorization_code&client_id=<YOUR_CLIENT_ID>&client_secret=<Y-
OUR_CLIENT_SECRET>&code={code}&<scope>=offline_access&redirect_uri=<Y-
OUR_REDIRECT_URI>
```

If the previous request was succesful, it will return a JSON object containing a pair of JWT (RFC 7519) access and refresh tokens which can be used to access Webfleet API on behalf of the user which granted access.

It also contains information about the token as specified by JSON Web Token specification (RFC 7519) which is the underlying claim transport mechanism.

JSON object response example:

```
{
    "access_token": "eyJhbGciO...",
    "token_type": "bearer",
    "refresh_token": "eyJhbGciOiJ...",
    "expires_in": 3599,
    "scope": "...",
    "jti": "..."
    ...
}
```

Both access and refresh tokens have an expiration time, access tokens are short-lived while refresh tokens are long-lived. The former are used to access APIs while the latter are only used to renew access tokens using this procedure. Usually only refresh token needs to be persisted while access token may expire oftenly while requesting a protected resource. This should be handled specifically to trigger an access token renewal procedure as described below.

Typically access tokens will only persist during an execution session and refresh tokens will be used to issue new ones for a later session, thus, it is only necessary to store refresh tokens.

Refresh tokens can be used to issue access tokens thus impersonating the access of the userto the data - keep it mind while handling refresh tokens.

It is your responsibility to securely store refresh tokens using appropriate encryption and security mechanisms. Any leak must be communicated and refresh tokens revoked as soon as possible.

Alternatively `client_id` and `client_secret` parameters can be replaced with an authorisation header using Basic Authentication.

Why is it necessary to use offline tokens?

It is suggested adding “offline_access” scope in the process of obtaining access tokens in order to get an offline token. It is an optional scope and will not be applied by default unless it is added directly in the request.

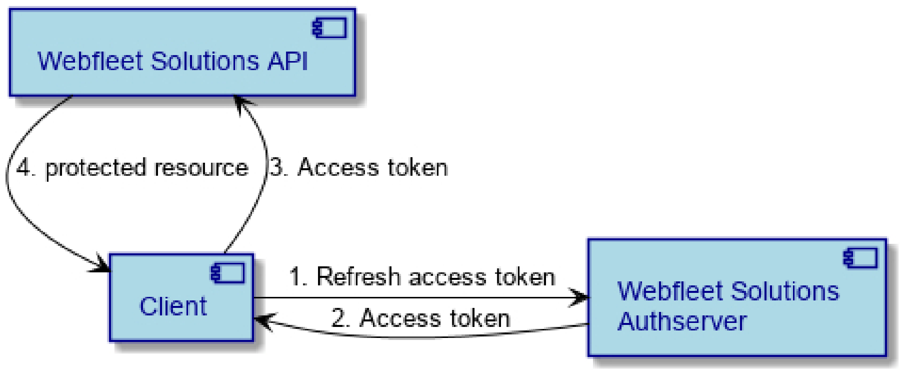
Offline access is a new feature implemented by our services that allows the application with an offline token to get the access token and use the resources without the user having to log in, for a long time or forever. In other words, users can grant access to the resources without having to introduce the credentials or, even, being present. Storing and handling of the offline token is done in the same way as with a standard refresh token.

This feature implies several improvements such as:

- It will allow to perform "offline" actions on behalf of the user even when the user is not online. For example, a periodic backup of some data every night.
- The offline token will never expire. However, it is subject to an idle timeout to renew its validity. It can also be revoked by the user at any time.

Refreshing access tokens

You can obtain a new access token as specified in OAuth 2.0 Refresh token using the token endpoint in a similar process as with the authorisation code to obtain an access token, OAuth client credentials are also required.



Refreshing an access token requires the following parameters in a form encoded request.

Parameter	Description
grant_type	OAuth grant type flow. Use `refresh_token` as parameter value.
refresh_token	The refresh token to use to renew the access token which was obtained during the initial flow authenticating the resource owner user.
client_id	OAuth client credentials username provided during registration process.
client_secret	OAuth client credentials password provided during registration process.

```
POST /auth/realms/webfleet/protocol/openid-connect/token HTTP/1.1
Host: https://login.webfleet.com
Content-type: application/x-www-form-urlencoded

grant_type=refresh_token&client_id=<YOUR_CLIENT_ID>&client_secret=<Y-
OUR_CLIENT_SECRET>&refresh_token=eyJhbGciOiJ...
```

A successful request will return for example:

```
{
    "access_token": "eyJhbGciO...",
    "token_type": "bearer",
    "refresh_token": "eyJhbGciOiJ...",
    "expires_in": 3599,
    "scope": "...",
    "jti": "..."
    ...
}
```

The `refresh_token` value from the JSON object response must be stored replacing the previously stored refresh token.

Alternatively `client_id` and `client_secret` parameters can be omitted and provided in an authorisation header using Basic Authentication.

Revoking refresh tokens

Revocation of refresh tokens is implemented following OAuth 2.0 Token revocation (RFC 7009). Given Webfleet Authorization Server uses JSON Web Token specification (RFC 7519) to issue signed self-contained tokens, only refresh tokens can be centrally revoked, access tokens stay valid until they have expired and cannot be revoked.

OAuth clients may revoke any refresh token issued to them, thus not requiring customer's consent to revoke access to a customer granted refresh token.

Revoking a refresh token requires the following parameters in a form encoded request.

Parameter	**Description**
token	Refresh token to revoke. Only refresh tokens are allowed for revocation.

Example using Basic Authentication

```
POST /auth/realms/webfleet/protocol/openid-connect/revoke HTTP/1.1
Host: https://login.webfleet.com
Authorization: Basic PHlvdXJfY2xpZW50X2lkPjo8eW91cl9jbGllbnRfc2VjcmV0PiA=
Content-type: application/x-www-form-urlencoded

token=eyJhbGciOiJ...
```

References

Please refer to specifications below for further details on the underlying authentication and authorization mechanisms.

- OAuth 2.0 (RFC 6749)
- OAuth 2.0 Bearer Token Usage (RFC 6750)
- JSON Web Tokens (RFC 7519)
- OAuth 2.0 Token revocation (RFC 7009)
- Hypertext Transfer Protocol (HTTP/1.1): Authentication (RFC 7235)
- Server Administration Guide – Offline access (keycloak.org)

Revision history

Revision history

Date	Description	Author
2020-05-12	Initial release	RH
2022-06-27	Updated content	TB/RH
2022-08-15	Added client_secret parameter to tables	TB/RH