



Authenticating Your REST API Client Using OAuth

The threat defense REST API uses OAuth 2.0 for authenticating calls from API clients. OAuth is an access token-based method, and the threat defense uses JSON web tokens for the schema. The relevant standards are:

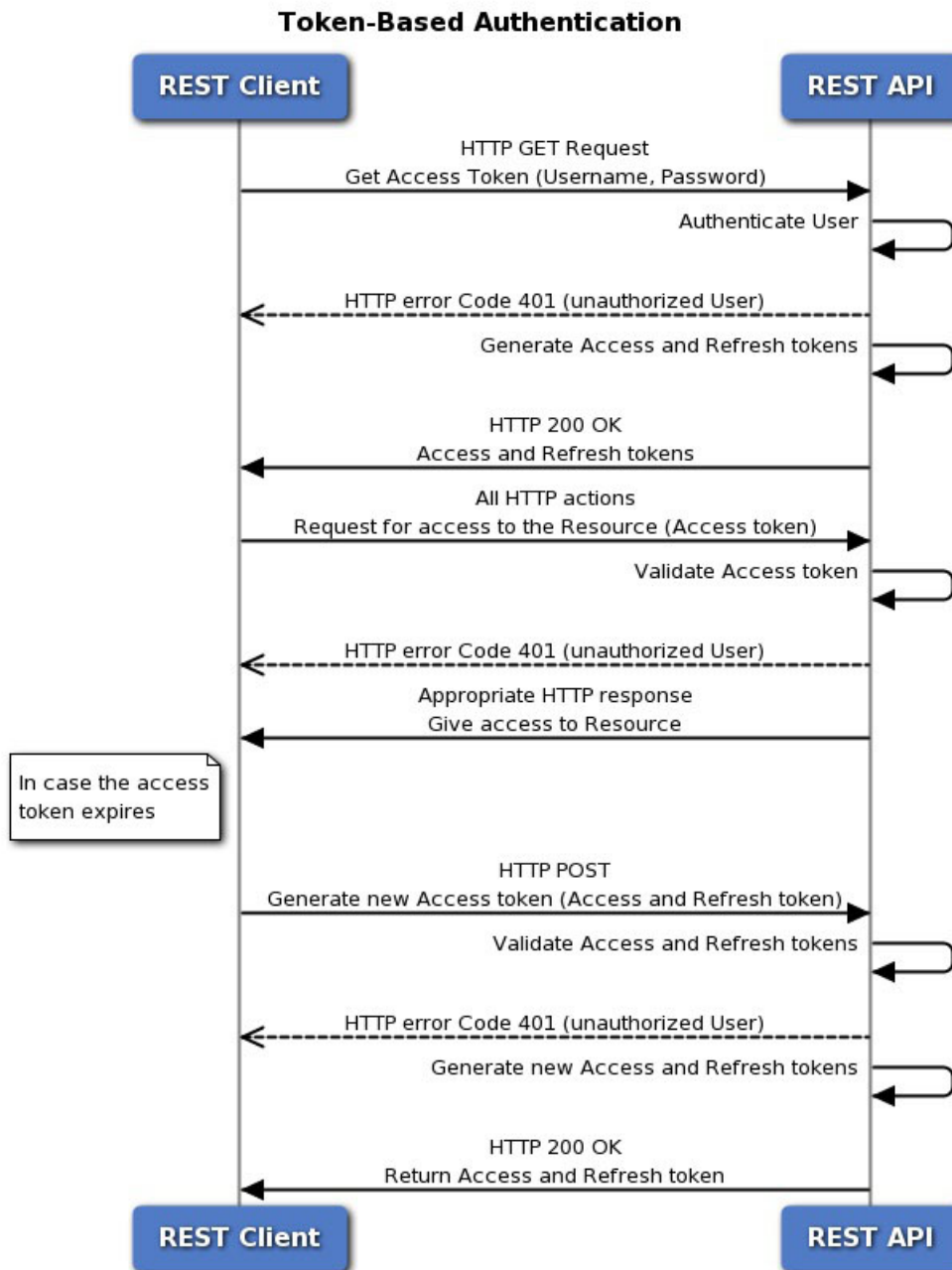
- RFC6749, The OAuth 2.0 Authorization Framework, <https://tools.ietf.org/html/rfc6749>.
- RFC7519, JSON Web Token (JWT), <https://tools.ietf.org/html/rfc7519>.

The following topics explain the methods for obtaining and using the required tokens.

- [Overview of the API Client Authentication Process, on page 1](#)
- [Requesting a Password-Granted Access Token, on page 3](#)
- [Requesting a Custom Access Token, on page 5](#)
- [Using an Access Token on API Calls, on page 7](#)
- [Refreshing an Access Token, on page 7](#)
- [Revoking an Access Token, on page 9](#)

Overview of the API Client Authentication Process

Following is the end-to-end view of how to authenticate your API client with the threat defense device.



Before you begin

Each token represents an HTTPS login session, which counts for API sessions and device manager sessions. There can be a maximum of 5 active HTTPS sessions. If you exceed this limit, the oldest session, either the device manager login or API token, is expired to allow the new session. Thus, it is important that you get only those tokens you need, and you reuse each token until it is expired, then renew them. Getting a new token for each API call will result in severe session churn and could lock users out of the device manager. These limits do not apply to SSH sessions.

Procedure

-
- Step 1** Authenticate the API client user using whatever method you require.
- Your client is obligated to authenticate users and ensure they have the authority to access and modify the threat defense device. If you want to provide differential abilities based on authorization rights, you need to build that into your client.
- For example, if you want to allow read-only access, you must set up the required authentication server, user accounts, and so forth. Then, when a user with read-only rights logs into your client, you must ensure that you issue GET calls only. In API v1, this type of variable access cannot be controlled by the threat defense device itself. Starting with API v2, if you are using external users and you do not groom calls based on user authorization, you will get errors if there is a mismatch between user authorization and the calls you attempt.
- For v1, when communicating with the device, you must use the **admin** user account on the threat defense device. The **admin** account has full read/write authorization for all user-configurable objects.
- Step 2** Request a password-granted access token based on username/password using the **admin** account.
- See [Requesting a Password-Granted Access Token, on page 3](#).
- Step 3** Optionally, request a custom access token for your client.
- With a custom token, you can explicitly request a validity period, and assign a subject name for the token. See [Requesting a Custom Access Token, on page 5](#).
- Step 4** Use the access token on API calls in the Authorization: Bearer header.
- See [Using an Access Token on API Calls, on page 7](#).
- Step 5** Before the access token expires, refresh the token.
- See [Refreshing an Access Token, on page 7](#).
- Step 6** When you are finished, revoke the token if it has not yet expired.
- See [Revoking an Access Token, on page 9](#).
-

Requesting a Password-Granted Access Token

Every REST API call must include an authentication token to verify that the caller is authorized to perform the requested action. Initially, you need to obtain an access token by supplying the **admin** username/password. This is called a password-granted access token, that is, `grant_type = password`.

Procedure

-
- Step 1** Create the JSON object for the password-granted access token grant.

```
{
```

```

    "grant_type": "password",
    "username": "string",
    "password": "string"
  }

```

Specify the **admin** username and the correct password, for example:

```

{
  "grant_type": "password",
  "username": "admin",
  "password": "Admin123"
}

```

Step 2 Use POST /fdm/token to obtain the access token.

For example, the **curl** command would look like the following:

```

curl -X POST --header 'Content-Type: application/json' --header
'Accept: application/json' -d '{
  "grant_type": "password",
  "username": "admin",
  "password": "Admin123"
}' 'https://ftd.example.com/api/fdm/latest/fdm/token'

```

Step 3 Retrieve the access and refresh tokens from the response.

A good response (status code 200) looks like the following:

```

{
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlMDI4MzI2NjcsInN1YiI6ImFkbWluIiwianRpIjoimGM3ZDBmNDgtODIwMS0xMWU3LWE4MWMtMDcwZmYzOWU3ZjQ0IiwibmJmIjoxNTAyODMyNjY3LCJleHAiOiJlMDI4MzQ0NjcsInJlZnJlc2hUa2t1bWV4cGlyZXNbdCI6MTUwMjgzNTA2NzQxOSwidG9rZW5UeXB1IjoislUX0FjY2VzcyIsIm9yaWdpbiI6InBhc3N3b3JkIn0.b2hI6fVA_GbmhCOPM-ZUx6IC8SgCk1AkHXI-llV0r7s",
  "expires_in": 1800,
  "token_type": "Bearer",
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlMDI4MzI2NjcsInN1YiI6ImFkbWluIiwianRpIjoimGM3ZDBmNDgtODIwMS0xMWU3LWE4MWMtMDcwZmYzOWU3ZjQ0IiwibmJmIjoxNTAyODMyNjY3LCJleHAiOiJlMDI4MzUwNjcsImFjY2VzclRva2VuRXhwaXJlc0F0IjoxNTAyODM0NDY3NDE5LCJyZWZyZXNoQ291bnQiOi0xLCJ0b2t1bWV4cGlyZXNbdCI6MTUwMjgzNTA2NzQxOSwidG9rZW5UeXB1IjoislUX0FjY2VzcyIsIm9yaWdpbiI6InBhc3N3b3JkIn0.iLNqz1c1X1vcq0j9pQYW4gwYsvUCcSyaidRXGutAz_o",
  "refresh_expires_in": 2400
}

```

Where:

- **access_token** is the bearer token you need to include on API calls. See [Using an Access Token on API Calls, on page 7](#).
- **expires_in** is the number of seconds for which the access token is valid, from the time the token is issued.
- **refresh_token** is the token you would use on a refresh request. See [Refreshing an Access Token, on page 7](#).

- **refresh_expires_in** is the number of seconds for which the refresh token is valid. This is always longer than the access token validity period.

Requesting a Custom Access Token

You can use the password-granted access token. However, you can also request a custom access token. With a custom token, you can supply a subject name to help differentiate token usage (for your own purposes). You can also request specific validity periods if the default values returned for password tokens do not fit your requirements.

Before you begin

You must first get a password-granted access token before getting a custom token. See [Requesting a Password-Granted Access Token, on page 3](#).

In addition:

- You can request a custom token only if you are a local user. External users cannot request custom tokens.
- You can use a custom token on the unit for which you obtain it only. You cannot use the token on the peer device in a high availability group.

Procedure

Step 1 Create the JSON object for the custom access token grant.

```
{
  "grant_type": "custom_token",
  "access_token": "string",
  "desired_expires_in": 0,
  "desired_refresh_expires_in": 0,
  "desired_subject": "string",
  "desired_refresh_count": 0
}
```

Where:

- **access_token** is a valid password-granted access token.
- **desired_expires_in** is an integer representing the number of seconds for which the custom access token will be valid. In comparison, the password-granted tokens are valid for 1800 seconds.
- **desired_refresh_expires_in** is an integer representing the number of seconds for which the custom refresh token will be valid. If you obtain a refresh token, ensure that this value is larger than the **desired_expires_in** value. In comparison, the password-granted refresh tokens are valid for 2400 seconds. This parameter is not required if you specify 0 for **desired_refresh_count**.
- **desired_subject** is a name you give to the custom token.


```
CiCSA",
  "refresh_expires_in": 3000
}
```

Where:

- **access_token** is the bearer token you need to include on API calls. See [Using an Access Token on API Calls, on page 7](#).
- **expires_in** is the number of seconds for which the access token is valid, from the time the token is issued.
- **refresh_token** is the token you would use on a refresh request. See [Refreshing an Access Token, on page 7](#).
- **refresh_expires_in** is the number of seconds for which the refresh token is valid. This is always longer than the access token validity period.

Using an Access Token on API Calls

After you obtain either a password-granted or custom access token, you must include it on each API call in the **Authorization: Bearer** header to the HTTPS request.

For example, a **curl** command to perform GET /object/networks might look like the following:

```
curl -k -X GET -H 'Accept: application/json'
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.yJp
YXQiojE1MDI4MzU5OTEsInN1YiI6ImFwaS1jbGllbnQiLCJqdG
kiOiJjOWIwYzYzdjYi04MjA4LTExZTctYTgxYy02YmY0NzY3ZmRm
ZGUuLCJuYmYiOiJ1MDI4MzU5OTEsImV4cCI6MTUwMjgzODM5MS
wiczVmcmVzaFRva2VuRXhwaXJlc0F0IjoxNTAyODM4OTkxMzIx
LCJ0b2t1b1R5cGUiOiJKV1RfQWNjZXNzIiwib3JpZ22luIjoieY3
VzdG9tIn0.9IVzLjGffvQffHAWdrNkrYfvu06TgpJ7Zi_z3RYu
bN8 '
'https://ftd.example.com/api/fdm/latest/object/networks'
```



Note

When you use the API Explorer to try out methods and resources, the **curl** command shown does not include the **Authorization: Bearer** header. However, you must add this header when making calls from your API client.

Refreshing an Access Token

After an access token expires, you need to refresh it using the refresh token that was supplied in the original grant. A refreshed access token is actually different than the original access token. “Refreshing” actually supplies a new pair of access token and refresh token, it does not simply extend the life of the old access token.

Procedure

Step 1

Create the JSON object for the refresh token grant.

```
{
  "grant_type": "refresh_token",
  "refresh_token": "string"
}
```

The **refresh_token** can be from a password-granted or custom access token grant.

For example:

```
{
  "grant_type": "refresh_token",
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlMDI0MzU0OTESInN1YiI6ImFwaSIjbGlblnQilCjQqdGk6IoiJyOWIxYzdjYi04MjA4LTExZTctYTg9Y0Y0ZmY0NzY3ZmRmZGUlLCJuYmYiOiJlMDI0MzU0OTESImV4cCI6MTUwMjgzODk5MSwiYWVudjkiZW50eHBlbnBvcmlnaW4iOiJkdXN0b20ifQ.qsejqj3Uo183YvfN_77iJZELEqwpWw5AbKAqaNCicSA"
}
```

Step 2 Use POST /fdm/token to obtain the refreshed access token.

For example, the **curl** command would look like the following:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{
  "grant_type": "refresh_token",
  "refresh_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlMDI0MzU0OTEsInN1YiI6ImFwaS1jbGllbnQiLCJqdGkiOiJ0eWlxeXZkdjYiO4MjA4LTE4TExzTctYTgxYy02YmY0NzY3ZmRmZGUlLCJuYmYiOiJlMDI0MzU0OTEsImV4cCI6MTUwMjgzODk5MSwiYWVjZXRmZG9rZW5FeHBpcmVzQXQiOiJlMDI0MzgzOTEzMEsInJlZnJlc2h0bD03VudCI6MywidG9rZW5UeXB1Ijois1dUX1JlZnJlc2giLCJvcmlnaW4iOiJjdXN0b20ifQ.qsejg3Uo183YvN_77iJZELBqwpWw5abiKqAqAnCtCSA"}' https://ftd.example.com/api/fdm/latest/fdm/token'
```

Step 3

Retrieve the access and refresh tokens from the response.

A good response (status code 200) looks like the following. In this example, the refresh token was for a custom token. The expiration periods are based on the values from the original custom access token request.

[illegible]

Where:

- ## Revoking an Access Token

Procedure

Where:

- 9

- (Do not use.) **custom_token_id_to_revoke** identifies custom access token by its internal unique ID. However, there is no direct way for you to obtain this value. Use the other options instead.
- **custom_token_subject_to_revoke** is the **desired_subject** value for the custom access token that you want to revoke.

For example:

```
{
  "grant_type": "revoke_token",
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlMDI5MDQzMjQsInN1YiI6ImFkbWluIiwianRpIjoizTMzNGIxOWYtODJhNy0xMWU3LWE4MWMtNGQ3NzY2ZTEzMzVkiwiwibmJmIjoxNTAyOTA0MzI0LCJleHAiOiJlMDI5MDYxMjQsInJlZnJlc2hUb2t1bkV4cGlyZXNbdCI6MTUwMjkwNjcyNDExMiwidG9rZW5UeXB1IjoislUX0FjY2VzcyIsIm9yaWdpbiI6InBhc3N3b3JkIn0.OVZBT9yVZc4zxZfZiiLH4SZcFclaHyCPbZJC_Gyd5FE",
  "custom_token_subject_to_revoke": "api-client"
}
```

Step 2 Use POST /fdm/token to revoke the access token.

For example, the **curl** command would look like the following:

```
curl -X POST --header 'Content-Type: application/json'
--header 'Accept: application/json' -d '{
  "grant_type": "revoke_token",
  "access_token": "eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOiJlMDI5MDQzMjQsInN1YiI6ImFkbWluIiwianRpIjoizTMzNGIxOWYtODJhNy0xMWU3LWE4MWMtNGQ3NzY2ZTEzMzVkiwiwibmJmIjoxNTAyOTA0MzI0LCJleHAiOiJlMDI5MDYxMjQsInJlZnJlc2hUb2t1bkV4cGlyZXNbdCI6MTUwMjkwNjcyNDExMiwidG9rZW5UeXB1IjoislUX0FjY2VzcyIsIm9yaWdpbiI6InBhc3N3b3JkIn0.OVZBT9yVZc4zxZfZiiLH4SZcFclaHyCPbZJC_Gyd5FE",
  "custom_token_subject_to_revoke": "api-client"
}' 'https://ftd.example.com/api/fdm/latest/fdm/token'
```

Step 3 Evaluate the response to verify that the token was revoked.

A good response (status code 200) looks like the following.

```
{
  "message": "OK",
  "status_code": 200
}
```