Web Authorization Protocol
Internet-Draft
Intended status: Informational
Expires: 20 April 2026

N. Watson
Google, LLC

17 October 2025

OAuth 2.0 Refresh Token and Authorization Expiration draft-watson-oauth-refresh-token-expiration-01

Abstract

This specification extends OAuth 2.0 [RFC6749] by adding new token endpoint response parameters to specify refresh token expiration and user authorization expiration.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://njwatson32.github.io/rt-expiration/draft-watson-oauth-refresh-token-expiration.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-watson-oauth-refresh-token-expiration/.

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (mailto:oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/. Subscribe at https://www.ietf.org/mailman/listinfo/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/njwatson32/rt-expiration.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 20 April 2026.

Table of Contents

1. Introduction

RFC6749 defines the OAuth 2.0 protocol, part of which is the ability for a client to receive a refresh token that may be repeatedly exchanged for more access tokens. OAuth 2.0 does not contain any normative language around expiration or lack thereof for refresh tokens, mentioning only that they are "typically long-lasting".

In the years since the publication of OAuth 2.0, in response to changing security and privacy landscapes, many authorization servers have begun to issue shorter-lived refresh tokens for two main reasons:

* The authorization server or user may decide that the access being granted is too sensitive to allow indefinite access (e.g. mail or health data).

* The authorization server enforces a maximum duration that refresh tokens may be held without being exchanged on the token endpoint.

Clients may wish to implement special handling for expiring refresh tokens. For example, if the user has granted expiring access, the client may notify the user that they will need to reauthorize access before a certain date to avoid interruption of service.

2. Requirements Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

"Resource owner" and "user" may be used interchangeably to refer to the entity capable of granting access to a protected resource.

"Client", "application", and "relying party" may be used interchangeably to refer to the application making protected resource requests on behalf of the resource owner and with its authorization.

4. Concepts

There are two mechanisms that can affect refresh token expiration.

4.1. Authorization expiration

When granting authorization for an application to access their data as referenced in [RFC6749] Sec 4.1.1, the user may opt to time-limit that authorization, especially if the data is sensitive or they aren't sure how long they'll continue using the application. The authorization server itself may also impose mandatory limits on authorization duration.

4.2. Refresh token timeout

Authorization servers may wish to define a maximum amount of time clients can hold a refresh token without exchanging it. Beyond the security benefit provided by expiring credentials, this also provides a convenient mechanism for authorization servers to ensure there aren't ancient valid credentials out in the wild, which could complicate tasks like refresh token key rotation.

5. Refresh token expiration

The refresh token MUST NOT expire later than the user authorization expires. It MAY expire earlier if the authorization server also enforces a maximum duration between refresh token exchanges.

If the user renews their authorization, the authorization server SHOULD update the expiration time of existing refresh tokens if their lifetime was truncated due to user authorization expiration. The authorization server MUST NOT accept expired refresh tokens for any purpose, even if it has no way to update the expiration time of existing refresh tokens.

Access tokens MUST NOT expire later than the user authorization expires. If the user renews their authorization, the authorization server MAY update the expiration time of existing access tokens if possible. Resource servers MUST NOT accept expired access tokens for any purpose, even if the authorization server has no way to update the expiration time of existing access tokens.

6. Token endpoint response

This specification introduces two new response parameters.

6.1. Successful response

refresh_token_timeout
: The time in seconds that the refresh token may be held by the client without exchanging. For example, the value 604800 denotes that the refresh token will expire in one week from the time the response was generated. This value SHALL NOT exceed the value in authorization_expires_in.

authorization_expires_in
: The lifetime in seconds of the user's authorization for the scopes contained in the issued or presented refresh token. For example, the value 2629800 denotes that the authorization will expire in one month from the time the response was generated. This value MAY exceed that of refresh_token_timeout.

If finite, the authorization server MUST return these values whenever the token endpoint response contains the refresh_token field. The authorization server MAY return these values even if the response contains no refresh_token field in the response, which can be useful in the following example cases:

* For refresh_token_timeout, the authorization server could have updated the existing refresh token lifetime in place.

* For authorization_expires_in, the user's authorization lifetime could have been modified out of band.

* In all cases, it can be convenient for the client to receive these values in each response.

6.1.1. Relationship of authorization_expires_in to scopes

Though authorization_expires_in is returned from the token endpoint when refresh tokens are used, it corresponds to the user's authorization for _scopes_ (or finer-grained access through RAR [RFC9396]) rather than individual tokens. The authorization server SHOULD ensure consistent lifetimes across multiple refresh tokens for the same scopes.

Tying authorization lifetime to scopes means it's possible to have some access valid for one duration and other access valid for a different duration. For example, a user could grant indefinite access for the openid scope and short-lived access for a calendar scope.

6.1.2. Infinite Expiration

Omitted values indicate that there is no fixed upper bound on the lifetime of the credential or authorization. If the authorization server has not declared its support for refresh token lifetime in the Authorization Server Metadata, omitted response fields could indicate either indefinite validity or simply lack of support for this specification. However, infinite expiration and lack of information about expiration should be handled by the client in the same way. That is to say, the client must always handle refresh token invalidation not caused by expiration, such as by explicit user revocation.

Rather than omitting a response value, an authorization server may choose to return a large arbitrary value, e.g. 315569520 for 10 years. This avoids any ambiguity around support for infinite values while achieving a similar practical effect. Clients MUST treat all large values as literals and MUST NOT make any assumptions about which may be considered infinite.

6.2. Error response

The existing invalid_grant error code already explicitly covers token expiration and should be sufficient. Upon receiving this error code the client SHOULD start a new authorization grant flow.

6.3. Example

Suppose an authorization server enforces that refresh tokens must be exchanged at least once every 7 days, and a user has granted authorization to an application for access for 30 days. The initial exchange will result in the following response values:

```
refresh_token_timeout: 604800  // 7 days
authorization_expires_in: 2592000  // 30 days
```

An exchange 7 days after initial authorization will result in the following response values:

```
refresh_token_timeout: 604800  // 7 days
authorization_expires_in: 1987200  // 23 days
```

An exchange 28 days after initial authorization will result in the following response values:

```
refresh_token_timeout: 172800  // 2 days
authorization_expires_in: 172800  // 2 days
```

7. Update to Authorization Server Metadata

Support for the expiring refresh tokens SHOULD be declared in the OAuth 2.0 Authorization Server Metadata [RFC8414] with the following metadata:

refresh_token_expiration_types_supported
: OPTIONAL. JSON array of supported expiration types. The possible values are "authorization" and "credential".

If the authorization server omits expiration time response fields to indicate indefinite validity, it MUST declare refresh_token_expiration_types_supported in its metadata to indicate to the client that it’s aware of this spec.

8. User Experience Considerations

While clients must be able to gracefully handle tokens’ expiring at any time, the user experience may suffer if there’s an unintended interruption of service. This degradation of experience would most likely be felt by users of clients running in the background, such as task or travel management apps that rely on access to a user’s calendar or inbox.

If an application recognizes that its access is nearing expiration, it can proactively prompt the user for reauthorization next time they’re "in the loop" (e.g. using a parameter like prompt=consent from [OpenID]), or even communicate to the user out of band that their granted access is expiring.

9. Security Considerations

While it is possible to allow refresh token expiration to exceed that of user authorization expiration if the authorization server checks both timestamps when validating a refresh token, this is a potentially dangerous source of bugs in systems with complicated user authorization models. By requiring refresh tokens to expire no later than user authorization expires, there is less risk of bugs that accidentally provide data access to the client beyond the term of the user’s authorization.

Authorization servers implementing token rotation on every refresh [OAuth 2.1 Sec 4.3.1] may wish to enforce a maximum duration that a refresh token may be held without rotation, and this specification allows that duration to be communicated as part of the API rather than relying on documentation.

10. Privacy Considerations

Allowing users to time-limit their authorization is a privacy improvement. While this was already doable in regular OAuth implementations, the potential interruption of service for the user may have discouraged implementation of the feature. This specification provides a standardized way to mitigate that concern and should lead to greater adoption of time-limited authorization.

11. IANA Considerations

11.1. OAuth Parameters Registration

This specification registers the following OAuth parameter definitions in the IANA OAuth Parameters registry.

11.1.1. Registry Contents

* Name: refresh_token_timeout
  - Parameter Usage Location: token response
  - Change Controller: IETF
  - Reference: This document
* Name: authorization_expires_in
  - Parameter Usage Location: token response
  - Change Controller: IETF
  - Reference: This document

11.2. OAuth Authorization Server Metadata Registration

This specification registers the following Authorization Server Metadata definitions in the IANA OAuth Authorization Server Metadata registry.

11.2.1. Registry Contents

* Metadata Name: refresh_token_expiration_types_supported
  - Metadata Description: What types of refresh token expiration are supported by the authorization server
  - Change Controller: IETF

- Reference: This document

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/rfc/rfc8414>.

12.2. Informative References

[OpenID] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, <https://www.rfc-editor.org/rfc/rfc9396>.

Acknowledgments

TODO acknowledge.

Author’s Address

Nicholas Watson
Google, LLC
Email: nwatson@google.com