| | |
|---|---|
| Workgroup: | fapi |
| Internet-Draft: | fapi-grant-management-01 |
| Published: | 23 June 2021 |
| Intended Status: | Standards Track |
| Authors: | T. Lodderstedt | S. Low | D. Postnikov |
|---|---|---|---|
| | *yes.com* | *Biza.io* | |

# Grant Management for OAuth 2.0

## Abstract

This specification defines an extension of OAuth 2.0 [RFC6749] to allow clients to explicitly manage their grants with the authorization server.

# Table of Contents

# 1. Introduction

OAuth authorization servers issue access and refresh tokens based on privileges granted by a resource owner to a particular client in the course of an authorization process or based on pre-configured policies. The concept representing these privileges and their assignment to a particular client is sometimes referred to as "underlying grant".

Although this concept is fundamental to OAuth, there is no explicit representation of the grant in the OAuth protocol. This leads to the situation that clients cannot explicitly manage grants, e.g. query the status or revoke a grant that is no longer needed. The status is implicitly communicated if an access token refresh succeeds or fails or if an API call using an access token fails with HTTP status codes 401 (token is invalid) or 403 (token lacks privileges).

It also means the client cannot explicitly ask the authorization server to update a certain grant that is bound to a certain user. Instead the authorization server, typically, will determine a pre-existing grant using the client id from the authorization request and the user id of the authenticated resource owner.

If a client wants the authorization server to update a pre-existing grant, it needs to obtain identity data about the user and utilize it in a login hint kind of parameter to refer to the "same user as last time", exposing more identity data to the client than necessary.

Another pattern that was proposed is to use refresh tokens to refer to grants. This would require to send the refresh token as part of the authorization request through the front channel, which posses security issues since the refresh token is a credential and could leak and be injected that way.

There are also use cases in highly regulated sectors, e.g. Open Data, where the client might be forced to maintain concurrent, independent grants to represent the privileges delegated by the resource owner to this client in the context of a distinct service offered by the client to this user and using different client_ids is not appropriate.

In order to support the before mentioned use cases, this specification introduces a `grant_id` clients can use to identify a particular grant along with additional authorization request parameters to request creation and use of such grant ids. This specification also defines a new grant management API provided by the authorization server that clients can use to query the status of a grant and to revoke it.

### 1.1. Terminology

- Grant is a set of permissions (authorization) granted by a User to a Client. Grant is a resource captured and managed by an Authorization Server.
- Consent is a legal concept that can result in a grant being created, but also can include legal, audit, reporting, archiving and non-repudiation requirements. Grant is an authorization created as a result of consent.
- Grant Management API: a HTTP-based API provided by the authorization server that clients can use to query the status of, update, replace and revoke grants.
- Data Recipients (Australia and FDX) and TPPs (UK and PSD2) are examples of OAuth clients used to describe use cases below.
- Data Holders (Australia), ASPSPs (UK and PSD2) and Data Providers (FDX) are examples of OAuth Authorization Servers used to describe use cases below.

## 2. Overview

An authorization server supporting this extension allows a client to explicitly manage its grants. The basic design principle is that creation and update of grants is always requested using an OAuth authorization request while querying the status of a grant and revoking it is performed using the new Grant Management API.

The underlying assumption is that creation and updates of grants almost always require interaction with the resource owner. Moreover, the client is supposed to manage the grant ids along with the respective tokens on its own without support from the authorization server.

## 3. Use cases supported

### 3.1. Revoking a grant

A client needs an ability to revoke a particular grant.

Examples:

- In the UK, TPPs currently use `DELETE /account-access-consents/{ConsentId}` custom endpoint to revoke authorization on ASPSP side.

- In Australia, Data Recipients currently use `cdr_arrangement_id` and `POST /arrangements/revoke` custom endpoint to revoke authorization on Data Holder's side after a user revoked their consent via Data Recipient's dashboard.

Both could use standardized `grant_id` and grant management endpoint's `DELETE` operation to achieve the same.

## 3.2. Querying the details of a grant

There are a lot of business scenarios where some details of the grant could change post original authorisation. Therefore, clients might need a way to query the current details of a grant.

Examples:

- In the UK, TPPs currently use `GET /account-access-consents/{ConsentId}` custom endpoint to retrieve authorization details from the ASPSP.
- In banking, the client could query the details of a grant to determine what accounts have been added to the grant by a user or other fine grain details of the authorisation (when the user has a choice).
- In some scenarios, resource owner can be multiple natural persons, so additional authorisations might be required and this might occur after the original authorisation was granted by the intiating resource owner. The client can query the status of consent at any point after the authorization to determine if complete authorization has occured (by adhoc query or regular polling). Another scenario that fits in this category is multi-party approval process for business entities.
- Some jurisdictions require client's and authorisation server's applications to provide a dashboard to a user to view and revoke authorisations given to the authorisation server. Querying the details of the grant allows clients to have access to the up-to-date status and contents of the consent.

## 3.3. Replace the details of a grant

A client wants to replace existing privileges of a certain grant with the new privileges.

In some scenarios, clients might choose to replace the grant with the new one while keeping the same grant id. Old privileges will be revoked and new privileges will be added if approved by the user. The client has to specify full details of the new request.

Examples:

- In the UK and Australia, "replace" is supported when grant identifier is specified in the authorization request.

## 3.4. Update the details of a grant

A client wants to update details of the existing grant. Additional details are merged into the grant.

This is especially useful, if the client wants to add privileges as needed to an existing grant in order to be able to access APIs with the same access token or obtain new access token from a single refresh token. In some scenarios, clients might choose to update to just extend the duration of a grant.

The client only has to specify additional or amended authorisation details. The grant id will be kept the same.

The client might also have to start another authorization process if a certain API request fails due to missing privileges (typically a HTTP status code 403).

Examples that can be implemented using "update":

- Time extension of an authorisation.
- Other use cases that are covered OAuth Incremental Authorization (https://tools.ietf.org/html/draft-ietf-oauth-incremental-authz-04).

### 3.5. Support for concurrent grants

Some ecosystems allow multiple active authorisations between the same client, the same authorization server and the same resource owner at the same time (concurrent grants). In order to support concurrent grants, at a minimum, a client needs an ability to reference and revoke a particular grant, as well as, ability to create a new grant where there is an existing grant between the same parties.

Examples:

- In Australia, Data Recipients and Data Holders are mandated to support concurrent grants (authorizations). It's Data Recipient's choice to decide if a new grant is the replacement of a previous grant or a new grant.
- In UK, concurrent grants are also supported.

### 3.6. Creation of another resource

In some use cases, grant is a permission to create another resource. This created resource will have a separate lifecycle and can be managed outside of the Authorization Server.

Examples:

- For payment initiation, a new grant with a permission to create a payment request might be created first. A client can then use obtained access tokens to initiate the payment and, as a result, a new payment / transaction resource might be created.

### 3.7. Obtaining new tokens for existing grants

Clients can also obtain fresh access and, optionally refresh tokens based on existing grants if they re-issue authorization request, reference an existing grant and follow the rest of the authorization code flow.

## 4. Use cases not supported

### 4.1. Historical grant, authorisation or consent records

Grant Management specification allows a client to query the status and contents of a grant (resource owner's consent). This is designed for clients to understand what is included in current active grant. This is NOT designed to provide for legal, reporting or archiving purposes, for example, keeping 7 years of expired or revoked consents.

### 4.2. Consent resource shared with other parties

There is a use case where end user might want to share their consents with third parties (e.g. centralised consent management dashboards). A new Consent Resource API could be created for this purpose. This is out of scope for this specification.

There is also another use case where AS could support sharing of grants among clients belonging to a single administrative entity, for instance where an organisation delivers its service across different platforms (e.g. iOS, Android and a Web App each having separate clients). Sector identifier URIs as defined in [OpenID.Registration] is one option to achieve this objective. This is out of scope for this specification.

# 5. OAuth Protocol Extensions

## 5.1. Requirements for Authorization Servers

Grant management is restricted to confidential only clients due to security reasons.

## 5.2. Authorization Request

This specification introduces the authorization request parameters `grant_id` and `grant_management_action`. These parameters can be used with any request serving as authorization request, e.g. it may be used with CIBA requests.

`grant_id`: OPTIONAL. String value identifying an individual grant managed by a particular authorization server for a certain client and a certain resource owner. The `grant_id` value MUST have been issued by the respective authorization server and the respective client MUST be authorized to use the particular grant id.

`grant_management_action`: string value controlling the way the authorization server shall handle the grant when processing an authorization request. This specification defines the following values:

- `create`: the AS will create a fresh grant if the AS supports the grant management action `create`.
- `update`: this mode requires the client to specify a grant id using the `grant_id` parameter. If the parameter is present and the AS supports the grant management action `update`, the AS will merge the permissions consented by the user in the actual request with those which already exist within the grant.
- `replace`: this mode requires the client to specify a grant id using the `grant_id` paramter. If the parameter is present and the AS supports the grant management action `replace`, the AS will change the grant to be ONLY the permissions requested by the client and consented by the user in the actual request.

The following example shows how a client may ask the authorization request to use a certain grant id:

```
GET /authorize?response_type=code&
     client_id=s6BhdRkqt3
     &grant_management_action=update
     &grant_id=TSdqirmAxDa0_-DB_1bASQ
     &scope=write
     &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
     &code_challenge_method=S256
     &code_challenge=K2-ltc83acc4h... HTTP/1.1
Host: as.example.com
```

## 5.3. Authorization Error Response

In case the `grant_id` is unknown or invalid, the authorization server SHALL respond with an error code `invalid_grant_id`.

In case the AS does not support a grant management action requested by the client, or the grant management action is required (according to `grant_management_action_required` metadata) but not specified, SHALL respond with the error code `invalid_request`.

## 5.4. Token Response

This specification introduces the token response parameter `grant_id`:

`grant_id`: URL safe string value identifying an individual grant managed by a particular authorization server for a certain client and a certain resource owner. The `grant_id` value MUST be unique in the context of a certain authorization server and SHOULD have enough entropy to make it impractical to guess it.

The AS will return a `grant_id` if it supports any of the grant management actions `query`, `revoke`, `update`, `replace`.

Here is an example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-cache, no-store

{
   "access_token": "2YotnFZFEjr1zCsicMWpAA",
   "token_type": "example",
   "expires_in": 3600,
   "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
   "grant_id":"TSdqirmAxDa0_-DB_1bASQ"
}
```

## 5.5. Lifecycle of the grant

### 5.5.1. Creation

Grant, as a set of authorized permissions, is created by the AS on authorization request completion.

For the initial authorization flow, a grant should be considered active when associated tokens have been successfully claimed by the client.

If the tokens haven't been claimed the grant should be deleted by the AS after a reasonable timeout. Timeline of the deletion is left up to AS implementation.

### 5.5.2. Modification

Grant can be modified by a client via update or replace actions.

Some elements of grant can be updated by the AS to reflect the status of some resources included in the grant. For example, if a user chosen to share an account with a client and this account required additional authorisations before being considered as fully authorized.

### 5.5.3. Deletion

Authorization server may remove an obsolete grant at its discretion, but it should consider status and expiry of authorization elements included in the grant. The exact mechanism could differ between different deployments, for example, some deployments could purge a grant when all individual authorization_details attached to the grant have expired or revoked.

# 6. Grant Management API

The Grant Management API allows clients to perform various actions on a grant whose `grant_id` the client previously obtained from the authorization server.

Currently supported actions are:

- Query: Retrieve the current status of a specific grant
- Revoke: Request the revocation of a grant

The Grant Management API does not provide bulk access to all grants of a certain client for functional and privacy reasons. Every grant is associated with a certain resource owner, so just getting the status is useless for the client as long as there is not indication of the user the client can use this grant for. Adding user identity data to the status data would weaken the privacy protection OAuth offers for users towards a client.

The Grant Management API will not expose any tokens associated with a certain grant in order to prevent token leakage. The client is supposed to manage its grants along with the respective tokens and ensure its usage in the correct user context.

## 6.1. API authorization

Using the grant management API requires the client to obtain an access token authorized for this API. The grant type the client uses to obtain this access token is out of scope of this specification.

The token is required to be associated with the following scope value:

`grant_management_query`: scope value the client uses to request an access token to query the status of its grants.

`grant_management_revoke`: scope value the client uses to request an access token to revoke its grants.

## 6.2. Endpoint

The grant management API is provided by a new endpoint provided by the authorization server. The client MAY utilize the server metadata parameter `grant_management_endpoint` (see Section 7.1) to obtain the endpoint URL.

## 6.3. Grant Resource URL

The resource URL for a certain grant is built by concatenating the grant management endpoint URL, a slash, and the the `grant_id`. For example, if the grant management endpoint is defined as

```
https://as.example.com/grants
```

and the `grant_id` is

```
TSdqirmAxDa0_-DB_1bASQ
```

the resulting resource URL would be

```
https://as.example.com/grants/TSdqirmAxDa0_-DB_1bASQ
```

## 6.4. Query Status of a Grant

The status of a grant is queried by sending a HTTP GET request to the grant's resource URL as shown by the following example.

```
GET /grants/TSdqirmAxDa0_-DB_1bASQ
Host: as.example.com
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
```

The authorization server will respond with a JSON-formatted response as shown in the following example:

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store
Content-Type: application/json

{
   "scopes":[
      {
         "scope":"contacts read write",
         "resources":[
            "https://rs.example.com/api"
         ]
      },
      {
         "scope":"openid"
      }
   ],
   "claims":[
      "given_name",
      "nickname",
      "email",
      "email_verified"
   ],
   "authorization_details":[
      {
         "type":"account_information",
         "actions":[
            "list_accounts",
            "read_balances",
            "read_transactions"
         ],
         "locations":[
            "https://example.com/accounts"
         ]
      }
   ]
}
```

The privileges associated with the grant will be provided as a JSON array containing objects with the following structure:

- `scopes`: JSON array where every entry contains a `scope` field and may contain one or more `resource` fields. This structure allows the AS to represent the relationship between scope values and the resource indicators (as defined in [RFC8707]) they where requested and approved with. The concrete mapping is at the discretaion of the AS. The AS could, for example, organize those objects "by resource", i.e. for every resource there is a list of related scope values. It could also store chunks of scope values along with the resource parameter values as requested and approved in a certain authorization request.
- `claims`: JSON array containing the names of all OpenID Connect claims (see [OpenID]) as requested and consented in one or more authorization requests associated with the respective grant.

- `authorization_details`: JSON Object as defined in [I-D.ietf-oauth-rar] containing all authorization details as requested and consented in one or more authorization requests associated with the respective grant.

The response structure MAY also include further elements defined by extensions.

Where an OP is currently experiencing high load it may return an HTTP 503 with a Retry-After response header as described in Section 7.1.3 of [RFC7231]. Clients should respect such headers and only retry after the time indicated in the header.

### 6.5. Revoke Grant

To revoke a grant, the client sends a HTTP DELETE request to the grant's resource URL. The authorization server responds with a HTTP status code 204 and an empty response body.

This is illustrated by the following example.

```
DELETE /grants/TSdqirmAxDa0_-DB_1bASQ
Host: as.example.com
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA

HTTP/1.1 204 No Content
```

The AS MUST revoke the grant and all refresh tokens issued based on that particular grant, it SHOULD revoke all access tokens issued based on that particular grant.

Note: Token revocation as defined in [RFC7009] differentiates from grant revocation as defined in this specification in that token revocation is not required to cause the revocation of the underlying grant. It is at the discretion of the AS to retain a grant in case of token revocation and allow the client to re-connect to this grant through a subsequent authorization request. This decoupling may improve user experience in case the client just wanted to discard the token as a credential.

### 6.6. Error Responses

If the resource URL is unknown, the authorization server responds with HTTP status code 404.

If the client is not authorized to perform a call, the authorization server responds with HTTP status code 403.

If the request lacks a valid access token, the authorization server responds with HTTP status code 401 and invalid_token error code.

## 7. Metadata

### 7.1. Authorization server's metadata

`grant_management_actions_supported` OPTIONAL. JSON array containing the actions supported by the AS. Allowed values are `query`, `revoke`, `update`, `create`.

- `query`: the AS allows clients to query the permissions associated with a certain grant.
- `revoke`: the AS allows clients to revoke grants.
- `update`: the AS allows clients to update existing grants.

- `create`: the AS allows clients to request the creation of a new grant.

If omitted, the AS does not support any grant management actions.

`grant_management_endpoint` OPTIONAL. URL of the authorization server's Grant Management Administration Endpoint.

`grant_management_action_required` OPTIONAL. Boolean where, if `true` all authorization requests MUST specify a `grant_management_action`. If omitted defaults to `false`.

# 8. Implementation Considerations

## 8.1. Client to grant relationship

A client (as logical entity) MAY use multiple client ids to deliver its service across different platforms, e.g. apps for iOS and Android and a Web App. It is RECOMMENDED that the AS supports sharing of grants among client ids belonging to the same client. Sector identifier URIs as defined in [OpenID.Registration] is one option to group client ids under single administrative control.

## 8.2. Addressibility of grant components

Implementations may wish to consider solutions to allow for addressibility of individual components within a grant. Trust ecosystems should consider their requirements during implementation and consider either; - Including a unique identifier within the authorization object (ie. `id` within the RAR) or; - Defining a comparison algorithm for the grant to allow for derivation of update and append actions

# 9. Privacy Consideration

`grant_id` is issued by the authorization server for each established grant between a client and a user. This should prevent correlation between different clients.

It must not be possible to identify the user or derive any personally identifiable information (PII) based on `grant_id` alone.

`grant_id` potentially could be shared by different client_id belonging to the same entity.

# 10. Security Considerations

A grant id is considered a public identifier, it is not a secret. Implementations MUST assume grant ids leak to attackers, e.g. through authorization requests. For example, access to the sensitive data associated with a certain grant MUST NOT be made accessible without suitable security measures, e.g. an authentication and authorization of the respective client.

During the execution of a transaction utilising grant mode `replace` it is possible that the results of the resultant grant contain a permission set which is not a superset of the previous permission set. Consequently, where self contained access tokens are in use and there is a requirement for immediate propogation shorter than the lifespan of access tokens, the AS should immediately revoke all relevant tokens by an out-of-band means.

# 11. Normative References

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/info/rfc8707>.

[OpenID] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[I-D.ietf-oauth-rar] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", Work in Progress, Internet-Draft, draft-ietf-oauth-rar-04, 7 February 2021, <https://tools.ietf.org/html/draft-ietf-oauth-rar-04>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

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

## 12. Informative References

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <https://www.rfc-editor.org/info/rfc7009>.

[OpenID.Registration] Sakimura, N., Bradley, J., and M. Jones, "OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1", 8 November 2014, <https://openid.net/specs/openid-connect-registration-1_0.html>.

## Appendix A. IANA Considerations

`grant_id`

`grant_management_action`

`grant_management_actions_supported`

`grant_management_action_required`

`grant_management_endpoint`

`invalid_grant_id`

## Appendix B. Acknowledgements

We would like to thank Vladimir Dzhuvinov, Takahiko Kawasaki, Roland Hedberg, Filip Skokan, Dave Tonge, Brian Campbell and Ralph Bragg for their valuable feedback and contributions that helped to evolve this specification.

## Appendix C. Notices

## Appendix D. Document History

[[ To be removed from the final specification ]]

-01

- simplified authorization requests
- added metadata control grant management behavior of AS and client
- extended grant resource data model

-00

- initial revision

## Authors' Addresses

**Torsten Lodderstedt**
yes.com
Email: torsten@lodderstedt.net

**Stuart Low**
Biza.io
Email: stuart@biza.io

**Dima Postnikov**
Email: dima@postnikov.net