OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 4, 2020

W. Denniss
Google
May 3, 2020

OAuth 2.0 Incremental Authorization
draft-ietf-oauth-incremental-authz-04

Abstract

OAuth 2.0 authorization requests that include every scope the client might ever need can result in over-scoped authorization and a sub-optimal end-user consent experience. This specification enhances the OAuth 2.0 authorization protocol by adding incremental authorization, the ability to request specific authorization scopes as needed, when they’re needed, removing the requirement to request every possible scope that might be needed upfront.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

OAuth 2.0 clients may offer multiple features that require user authorization, but commonly not every user will use each feature. Without incremental authentication, applications need to either request all the possible scopes they need upfront, potentially resulting in a bad user experience, or track each authorization grant separately, complicating development.

The goal of incremental authorization is to allow clients to request just the scopes they need, when they need them, while allowing them to store a single authorization grant for the user that contains the sum of the scopes granted. Thus, each new authorization request increments the scope of the authorization grant, without the client needing to track a separate authorization grant for each group of scopes.

2. Notational 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 Key words for use in RFCs to Indicate Requirement Levels [RFC2119]. If these words are used without being spelled in uppercase then they are to be interpreted with their normal natural language meanings.

3. Terminology

In addition to the terms defined in referenced specifications, this document uses the following terms:

"OAuth" In this document, OAuth refers to OAuth 2.0 [RFC6749].

4. Incremental Auth for Confidential Clients

For confidential clients, such as web servers that can keep secrets, the authorization endpoint SHOULD treat scopes that the user already granted differently on the consent user interface. Typically such scopes are hidden for new authorization requests, or at least there is an indication that the user already approved them.

By itself, this property of the authorization endpoint enables incremental authorization. The client can track every scope they’ve ever requested, and include those scopes on every new authorization request.

To avoid the need for confidential clients to re-request already authorized scopes, authorization servers MAY support an additional "include_granted_scopes" parameter in the authorization request. This parameter, enables the client to request tokens during the authorization grant exchange that represent the full scope of the user’s grant to the application including any previous grants, without the client needing to track the scopes directly.

The client indicates they wish the new authorization grant to include previously granted scopes by sending the following additional parameter in the OAuth 2.0 Authorization Request (Section 4.1.1 of [RFC6749].) using the following additional parameter:

include_granted_scopes OPTIONAL. Either "true" or "false". When "true", the authorization server SHOULD include previously granted scopes for this client in the new authorization grant.

5. Incremental Auth for Public Clients

Unlike with confidential clients, it is NOT RECOMMEND to automatically approve OAuth requests for public clients without user consent (see Section 10.2 of OAuth 2.0 [RFC6749], and Section 8.6 of OAuth 2.0 [RFC8252]), thus authorization grants shouldn’t contain previously authorized scopes in the manner described above for confidential clients.

Public clients (and confidential clients using this technique) should instead track the scopes for every authorization grant, and only request yet to be granted scopes during incremental authorization. In the past, this would result in multiple discrete authorization grants that would need to be tracked. To enable incrementing a single authorization grant for public clients, the client supplies their existing refresh token during the authorization code exchange, and receives new authorization tokens with the scope of the previous and current authorization grants.

The client sends the previous refresh token in the OAuth 2.0 Access Token Request (Section 4.1.3 of [RFC6749].) using the following additional parameter:

existing_grant OPTIONAL. The refresh token from the existing authorization grant.

When processing the token exchange, in addition to the normal processing of such a request, the token endpoint MUST verify that token provided in the "existing_grant" parameter is unexpired and unrevoked, and was issued to the same client id and relates to the same user as the current authorization grant. If this verification succeeds, the new access and refresh tokens issued in the Access Token Response (Section 4.1.4 of) MUST include authorization for the scopes in the previous grant, unless the authorization server is exercising its prerogative to "fully or partially ignore the scope requested by the client" per Section 3.3 of OAuth 2.0 [RFC6749].

6. Usability Considerations

6.1. Handling Denials

A core principle of OAuth is that users may deny authorization requests for any reason. This remains true for incremental authorization requests. In the case of incremental authorization, clients may already have a valid authorization and receive a denial for an incremental authorization request (that is, an "access_denied" error code as defined in Section 4.1.2.1 of OAuth 2.0 [RFC6749]). Clients should SHOULD handle such errors gracefully and not discard any existing authorization grants if the user denies an incremental authorization request. Clients SHOULD NOT immediately request the same incremental authorization again, as this may result in an infinite denial loop (and the end-user feeling badgered).

6.2. Handling Scope Reductions

As specified by Section 5.1 of OAuth 2.0 [RFC6749], a successful response may not always include all the scope that was asked for, a fact indicated by the "scope" response parameter when it happens. This is still true in the case of incremental auth. The success response may include less scope than what was requested, or even less scope than before the incremental authorization request (say, if the user was given an opportunity to revise the grant down). Clients MUST check for the "scope" parameter in success responses and react accordingly.

For the purposes of an incremental auth request, a success response to an incremental authorization request that contains the same scope granted prior to the request being made, and an error response (for example, in the case of a denial) can have the same effect: the client retains a grant with the same scope as before. In the case of the approved request but with the same scope, they have a new grant, but with the same scope. In the case of the denied incremental authorization request, they still have the old grant with the same scope (although in some cases it may have been revoked or reduced in scope out of band).

An incremental authorization request isn’t the only time that scope can be reduced for a grant. As specified by section 6 of OAuth 2.0 [RFC6749], scope can be reduced during a token refresh as well. So it’s a good practice for clients to retain the current scope of the grant, update it during authorization, incremental authorization and token refreshes, and take action at any time based on the current scope by presenting an incremental authorization if a non-present scope is needed.

7. Alternative Approaches

This non-normative section discusses some alternative ways to achieve the incremental authorization result purely on the client side. These options are somewhat more complex and burdensome to client developers.

7.1. Alternative for Public Clients

It is possible for OAuth clients to maintain multiple authorizations per user for feature-specific scopes without needing the feature documented in this specification. For example, a public client (such as a mobile app) could maintain an authorization for the contacts and one for calendar, and store them separately.

This specification offers a convenience that a single authorization grant can be managed that represents all the scope granted so far, rather than needing to maintain multiple, however it does require that all grants are made from a single end-user account (as authorization servers cannot typically combine grants from multiple users). Clients where users may wish to authorize separate end-user accounts for different features should consider using the alternative documented in this section.

7.2. Alternative for Confidential Clients

An alternative incremental auth design for confidential clients is to ask for authorization scopes as they are needed and keep a running record of all granted scopes. In this way each incremental authorization request would include all scopes granted so far, plus the new scope needed. Authorization servers can see the existing scopes and only display the new scopes for approval (and likely to inform the user of the existing grants). This approach can be performed using RFC 6749 without additions, but requires the client to keep track of every authorization grant.

Confidential clients can also use the alternative documented for public clients in Section 7.1.

8. Privacy Considerations

8.1. Requesting Authorization In Context

The goal of incremental authorization is to enhance end-user privacy by allowing clients to request only the authorization scopes needed in the context of a particular user action, rather than asking for ever possible scope upfront. For example, an app may offer calendar and contacts integration, and an extension of OAuth like OpenID Connect for sign-in. Such an app should first sign the user in with just the scopes needed for that. If later the user interacts with the calendar or contacts features then, and only then, should the requires scopes be requested. By using this specification, apps can improve the privacy choices of end-users by only requesting the scopes they need in context.

Clients authorizing the user with an authorization server that supports incremental auth SHOULD ask for the minimal authorization scope for the user's current context, and use this specification to add authorization scope as required.

8.2. Preventing Overbroad Authorization Requests

When this specification is implemented, clients should have no technical reason to make overbroad authorization requests (i.e. requesting every possible scope they might ever expect need, rather than ones related to the user's current activity). To improve privacy, it is therefore RECOMMENDED for authorization servers to limit the authorization scope that can be requested in a single authorization to what would reasonably be needed by a single feature. The authorization server MAY deny such authorization requests with the following error code.

overbroad_scope
The scope of the request is considered overbroad by the authorization server. Consult the documentation of your authorization server to determine acceptable scope combinations, and consider using [[This Specification]] to perform incremental authorization requests in the context that the scope is needed.

Determining what constitutes an overbroad request is the purview of the authorization server. As an example, say an authorization supported "calendar" and "mail" scopes to access a user's calendar and inbox respectively. They may decide that their users should have the chance to grant such requests in context through incremental authorization, rather then all at once upfront, and deny the request for being overly broad.

8.3. Authorization Correlation

Incremental authorization is designed for use-cases where it's the same user authorizing each request, and thus all incremental authorization grants are correlated to that one user (by being merged into a single authorization grant). For applications where users may wish to connect different user accounts for different features (e.g. contacts from one account, and calendar from another) it is RECOMMENDED to instead allow multiple unrelated authorizations, as documented in Section 7.1.

The goal of this specification is to improve end-user privacy by giving them more choice over which scopes they grant access to. Previously many apps would request an overly large number of scopes upfront (typically for all the features of the app, rather than the subset that the user is currently wishing to use). The scopes in such authorization grants are necessarily correlated with the same user as they are contained in a single authorization grant. Implementing this specification doesn’t change that attribute, but it does improve user privacy overall by empowering the user to grant access in a more granular way.

8.4. Previously Granted Scopes

When the authorization server displays the list of scopes on page and prompts the user to consent to sharing access, users may assume that the displayed list of scopes on such a page is the full and complete list being granted to the application. It may be desirable for such a consent page to list previously granted scopes, provided that the client is confidential, or one that cannot be impersonated.

9. Discovery Metadata

Support for the incremental authorization MAY be declared in the OAuth 2.0 Authorization Server Metadata [RFC8414] with the following metadata:

incremental_authz_types_supported
OPTIONAL. JSON array of OAuth 2.0 client types that are supported for incremental authorization. The possible types are "confidential", and "public".

Specifically, "confidential" indicates that the behavior documented in Section 4 (Incremental Auth for Confidential Clients) is supported, and "public" indicates that the behavior documented in Section 5 (Incremental Auth for Public Clients) is supported.

A server which supports both forms of incremental auth documented in this specification would declare support like so:

```
"incremental_authz_types_supported": ["confidential", "public"]
```

10. Security Considerations

10.1. Public Client Impersonation

As documented in Section 8.6 of RFC 8252 [RFC8252], some public clients are susceptible to client impersonation, depending on the type of redirect URI used. If the "include_granted_scopes" feature documented in Section 4 is used by an impersonating client, it may receive a greater authorization grant than the user specifically approved for that client. For this reason, the "include_granted_scopes" feature MUST NOT be enabled for such public client requests.

Note that there is no such restriction on the use of "existing_grant" feature documented in Section 5. While it is designed for public clients, it MAY be supported for all client types.

11. IANA Considerations

This specification makes a registration request as follows:

11.1. OAuth Parameters Registry

This specification registers the following parameters in the IANA OAuth Parameters registry defined in OAuth 2.0 [RFC6749].

- Parameter name: include_granted_scopes

- Parameter usage location: authorization request

- Change controller: IESG

- Specification document(s): this document

- Parameter name: existing_grant

- Parameter usage location: token request

- Change controller: IESG

- Specification document(s): this document

11.2. OAuth Extensions Error Registration

This specification registers the following values in the IANA "OAuth Extensions Error Registry" registry [IANA.OAuth.Parameters] established by [RFC6749].

Name: overbroad_scope
Usage Location: authorization code grant error response
Protocol Extension: [[This Specification]]
Change Controller: IETF
Reference: Section 8.2 of [[This Specification]]

11.3. OAuth 2.0 Authorization Server Metadata

This specification registers the following values in the IANA "OAuth 2.0 Authorization Server Metadata" registry [IANA.OAuth.Parameters] established by [RFC8414].

- Metadata Name: incremental_authz_types_supported

- Metadata Description: JSON array containing a list of client types that support OAuth 2.0 Incremental Authorization [[this specification]]. The possible types are "confidential", and "public".
- Change controller: IESG
- Specification Document: Section 9 of [[this specification]]

12. 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/info/rfc2119>.

[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>.

[RFC8252] Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, <https://www.rfc-editor.org/info/rfc8252>.

[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/info/rfc8414>.

[IANA.OAuth.Parameters]
IANA, "OAuth Parameters", <http://www.iana.org/assignments/oauth-parameters>.

Appendix A. Acknowledgements

This document was produced in the OAuth working group under the chairpersonship of Rifaat Shekh-Yusef and Hannes Tschofenig with Benjamin Kaduk, and Eric Rescorla serving as Security Area Directors.

The following individuals contributed ideas, feedback, and wording that shaped and formed the final specification:

Yanna Wu, Marius Scurtescu, Jason Huang, Nicholas Watson, Breno de Medeiros, Naveen Agarwal, Brian Campbell, and Aaron Parecki.

Appendix B. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

01

- Changed a SHOULD to a MUST in Section 5 regarding the protocol behavior of incremental auth for public clients, while clarifying that the authorization server retains the prerogative to do whatever it wants.

- Defined an OAuth Metadata entry.

00

- Now a working group draft.

draft-wdenniss-oauth-incremental-auth-01

- Added usability, privacy, and security considerations.

- Documented alternative approaches.

draft-wdenniss-oauth-incremental-auth-00

- Initial draft based on the implementation of incremental and "appcremental" auth at Google.

Author’s Address

William Denniss
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
USA

Email: wdenniss@google.com
URI: https://wdenniss.com/incremental-auth