Stream:	Internet Engineering Task Force (IETF)
RFC:	8707
Category:	Standards Track
Published:	February 2020
ISSN:	2070-1721
Authors:	B. Campbell	J. Bradley	H. Tschofenig
---	---	---	---
	Ping Identity	*Yubico*	*Arm Limited*

RFC 8707
Resource Indicators for OAuth 2.0

Abstract

This document specifies an extension to the OAuth 2.0 Authorization Framework defining request parameters that enable a client to explicitly signal to an authorization server about the identity of the protected resource(s) to which it is requesting access.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8707.

Table of Contents

1. Introduction

Several years of deployment and implementation experience with the OAuth 2.0 Authorization Framework [RFC6749] has uncovered a need (in some circumstances, such as an authorization server servicing a significant number of diverse resources) for the client to explicitly signal to the authorization server where it intends to use the access token it is requesting.

Knowing the protected resource (a.k.a. resource server, application, API, etc.) that will process the access token enables the authorization server to construct the token as necessary for that entity. Properly encrypting the token (or content within the token) to a particular resource, for example, requires knowing which resource will receive and decrypt the token. Furthermore,

various resources oftentimes have different requirements with respect to the data contained in (or referenced by) the token, and knowing the resource where the client intends to use the token allows the authorization server to mint the token accordingly.

Specific knowledge of the intended recipient(s) of the access token also helps facilitate improved security characteristics of the token itself. Bearer tokens, currently the most commonly utilized type of OAuth access token, allow any party in possession of a token to get access to the associated resources. To prevent misuse, several important security assumptions must hold, one of which is that an access token must only be valid for use at a specific protected resource and for a specific scope of access. Section 5.2 of [RFC6750], for example, prescribes including the token's intended recipients within the token to prevent token redirect. When the authorization server is informed of the resource that will process the access token, it can restrict the intended audience of that token to the given resource such that the token cannot be used successfully at other resources.

OAuth scope, from Section 3.3 of [RFC6749], is sometimes overloaded to convey the location or identity of the protected resource, however, doing so isn't always feasible or desirable. Scope is typically about what access is being requested rather than where that access will be redeemed (e.g., `email`, `admin:org`, `user_photos`, `channels:read`, and `channels:write` are a small sample of scope values in use in the wild that convey only the type of access and not the location or identity).

In some circumstances and for some deployments, a means for the client to signal to the authorization server where it intends to use the access token it's requesting is important and useful. A number of implementations and deployments of OAuth 2.0 have already employed proprietary parameters toward that end. Going forward, this specification aspires to provide a standardized and interoperable alternative to the proprietary approaches.

1.1. 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.

1.2. Terminology

This specification uses the terms "access token", "refresh token", "authorization server", "resource server", "authorization endpoint", "authorization request", "authorization response", "token endpoint", "grant type", "access token request", "access token response", and "client" defined by The OAuth 2.0 Authorization Framework [RFC6749].

2. Resource Parameter

In requests to the authorization server, a client **MAY** indicate the protected resource (a.k.a. resource server, application, API, etc.) to which it is requesting access by including the following parameter in the request.

resource
: Indicates the target service or resource to which access is being requested. Its value **MUST** be an absolute URI, as specified by Section 4.3 of [RFC3986]. The URI **MUST NOT** include a fragment component. It **SHOULD NOT** include a query component, but it is recognized that there are cases that make a query component a useful and necessary part of the resource parameter, such as when one or more query parameters are used to scope requests to an application. The `resource` parameter URI value is an identifier representing the identity of the resource, which **MAY** be a locator that corresponds to a network-addressable location where the target resource is hosted. Multiple `resource` parameters **MAY** be used to indicate that the requested token is intended to be used at multiple resources.

The parameter value identifies a resource to which the client is requesting access. The parameter can carry the location of a protected resource, typically as an https URL or a more abstract identifier. This enables the authorization server to apply policy as appropriate for the resource, such as determining the type and content of tokens to be issued, if and how tokens are encrypted, and applying appropriate audience restrictions.

The client **SHOULD** provide the most specific URI that it can for the complete API or set of resources it intends to access. In practice, a client will know a base URI for the application or resource that it interacts with, which is appropriate to use as the value of the `resource` parameter. The client **SHOULD** use the base URI of the API as the `resource` parameter value unless specific knowledge of the resource dictates otherwise. For example, the value `https://api.example.com/` would be used for a resource that is the exclusive application on that host; however, if the resource is one of many applications on that host, something like `https://api.example.com/app/` would be used as a more specific value. Another example is when an API has multiple endpoints, e.g., when System for Cross-domain Identity Management (SCIM) [RFC7644] has endpoints such as `https://apps.example.com/scim/Users`, `https://apps.example.com/scim/Groups`, and `https://apps.example.com/scim/Schemas`. The client would use `https://apps.example.com/scim/` as the resource so that the issued access token is valid for all the endpoints of the SCIM API.

The following error code is provided for an authorization server to indicate problems with the requested resource(s) in response to an authorization request or access token request. It can also be used to inform the client that it has requested an invalid combination of resource and scope.

invalid_target
: The requested resource is invalid, missing, unknown, or malformed.

The authorization server **SHOULD** audience-restrict issued access tokens to the resource(s) indicated by the `resource` parameter. Audience restrictions can be communicated in JSON Web Tokens [RFC7519] with the `aud` claim and the top-level member of the same name provides the audience restriction information in a Token Introspection [RFC7662] response. The authorization server may use the exact `resource` value as the audience or it may map from that value to a more general URI or abstract identifier for the given resource.

2.1. Authorization Request

When the `resource` parameter is used in an authorization request to the authorization endpoint, it indicates the identity of the protected resource(s) to which access is being requested. When an access token will be returned directly from the authorization endpoint via the implicit flow (Section 4.2 of OAuth 2.0 [RFC6749]), the requested resource is applicable to that access token. In the code flow (Section 4.1 of OAuth 2.0 [RFC6749]) where an intermediate representation of the authorization grant (the authorization code) is returned from the authorization endpoint, the requested resource is applicable to the full authorization grant.

For an authorization request sent as a JSON Web Token (JWT), such as when using the JWT Secured Authorization Request [JWT-SAR], a single `resource` parameter value is represented as a JSON string while multiple values are represented as an array of strings.

If the client omits the `resource` parameter when requesting authorization, the authorization server **MAY** process the request with no specific resource or by using a predefined default resource value. Alternatively, the authorization server **MAY** require clients to specify the resource(s) they intend to access and **MAY** fail requests that omit the parameter with an `invalid_target` error. The authorization server might use this data to inform the user about the resources the client is going to access on their behalf, to apply policy (e.g., refuse the request due to unknown resources), and to determine the set of resources that can be used in subsequent access token requests.

If the authorization server fails to parse the provided value(s) or does not consider the resource (s) acceptable, it should reject the request with an error response using the error code `invalid_target` as the value of the `error` parameter and can provide additional information regarding the reasons for the error using the `error_description`.

An example of an authorization request where the client tells the authorization server that it wants an access token for use at `https://api.example.com/app/` is shown in Figure 1 below (extra line breaks and indentation are for display purposes only).

```
GET /as/authorization.oauth2?response_type=token
   &client_id=example-client
   &state=XzZaJlcwYew1u0QBrRv_Gw
   &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
   &resource=https%3A%2F%2Fapi.example.com%2Fapp%2F HTTP/1.1
Host: authorization-server.example.com
```

Figure 1: Implicit Flow Authorization Request

Below in Figure 2 is an example of an authorization request using the `code` response type where the client is requesting access to the resource owner's contacts and calendar data at `https://cal.example.com/` and `https://contacts.example.com/` (extra line breaks and indentation are for display purposes only).

```
GET /as/authorization.oauth2?response_type=code
   &client_id=s6BhdRkqt3
   &state=tNwzQ87pC6llebpmac_IDeeq-mCR2wLDYljHUZUAWuI
   &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
   &scope=calendar%20contacts
   &resource=https%3A%2F%2Fcal.example.com%2F
   &resource=https%3A%2F%2Fcontacts.example.com%2F HTTP/1.1
Host: authorization-server.example.com
```

Figure 2: Code Flow Authorization Request

2.2. Access Token Request

When the `resource` parameter is used on an access token request made to the token endpoint, for all grant types, it indicates the target service or protected resource where the client intends to use the requested access token.

The resource value(s) that is acceptable to an authorization server in fulfilling an access token request is at its sole discretion based on local policy or configuration. In the case of a `refresh_token` or `authorization_code` grant type request, such policy may limit the acceptable resources to those that were originally granted by the resource owner or a subset thereof. In the `authorization_code` case where the requested resources are a subset of the set of resources originally granted, the authorization server will issue an access token based on that subset of requested resources, whereas any refresh token that is returned is bound to the full original grant.

When requesting a token, the client can indicate the desired target service(s) where it intends to use that token by way of the `resource` parameter and can indicate the desired scope of the requested token using the `scope` parameter. The semantics of such a request are that the client is asking for a token with the requested scope that is usable at all the requested target services. Effectively, the requested access rights of the token are the cartesian product of all the scopes at all the target services. To the extent possible, when issuing access tokens, the authorization server should downscope the scope value associated with an access token to the value the respective resource is able to process and needs to know. (Here, "downscope" means give fewer permissions than originally authorized by the resource owner.) This further improves privacy as a list of scope values is an indication that the resource owner uses the multiple various services listed; downscoping a token to only that which is needed for a particular service can limit the extent to which such information is revealed across different services. As specified in Section 5.1 of [RFC6749], the authorization server must indicate the access token's effective scope to the client in the `scope` response parameter value when it differs from the scope requested by the client.

Following from the code flow authorization request shown in Figure 2, the below examples show an `authorization_code` grant type access token request (Figure 3) and response (Figure 4) where the client tells the authorization server that it wants the access token for use at `https://cal.example.com/` (extra line breaks and indentation are for display purposes only).

```
POST /as/token.oauth2 HTTP/1.1
Host: authorization-server.example.com
Authorization: Basic czZCaGRSa3F0Mzpoc3FFelFsVW9IQUU5cHg0RlNyNHlJ
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
&code=10esc29BWC2qZB0acc9v8zAv9ltc2pko105tQauZ
&resource=https%3A%2F%2Fcal.example.com%2F
```

Figure 3: Access Token Request

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

{
   "access_token":"eyJhbGciOiJFUzI1NiIsImtpZCI6Ijc3In0.eyJpc3MiOi
    JodHRwOi8vYXV0aG9yaXphdGlvbi1zZXJ2ZXIuZXhhbXBsZS5jb20iLCJzdWI
    iOiJfX2JfYyIsImV4cCI6MTU4ODQyMDgwMCwic2NvcGUiOiJjYWxlbmRhciIs
    ImF1ZCI6Imh0dHBzOi8vY2FsLmV4YW1wbGUuY29tLyJ9.nNWJ2dXSxaDRdMUK
    lzs-cYIj8MDoM6Gy7pf_sKrLGsAFf1C2bDhB60DQfW1DZL5npdko1_Mmk5sUf
    zkiQNVpYw",
   "token_type":"Bearer",
   "expires_in":3600,
   "refresh_token":"4LTC8lb0acc6Oy4esc1Nk9BWC0imAwH7kic16BDC2",
   "scope":"calendar"
}
```

Figure 4: Access Token Response

A subsequent access token request, using the refresh token, where the client tells the authorization server that it wants an access token for use at `https://contacts.example.com/` is shown in Figure 5 below with the response shown in Figure 6 (extra line breaks and indentation are for display purposes only).

```
POST /as/token.oauth2 HTTP/1.1
Host: authorization-server.example.com
Authorization: Basic czZCaGRSa3F0Mzpoc3FFelFsVW9IQUU5cHg0RlNyNHlJ
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token
&refresh_token=4LTC8lb0acc6Oy4esc1Nk9BWC0imAwH7kic16BDC2
&resource=https%3A%2F%2Fcontacts.example.com%2F
```

Figure 5: Access Token Request

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

{
   "access_token":"eyJhbGciOiJFUzI1NiIsImtpZCI6Ijc3In0.eyJpc3MiOi
    JodHRwOi8vYXV0aG9yaXphdGlvbi1zZXJ2ZXIuZXhhbXBsZS5jb20iLCJzdWI
    iOiJfX2JfYyIsImV4cCI6MTU4ODQyMDgyNiwic2NvcGUiOiJjb250YWN0cyIs
    ImF1ZCI6Imh0dHBzOi8vY29udGFjdHMuZXhhbXBsZS5jb20vIn0.5f4yhqazc
    OSlJw4y94KPeWNEFQqj2cfeO8x4hr3YbHtIl3nQXnBMw5wREY5O1YbZED-GfH
    UowfmtNaA5EikYAw",
   "token_type":"Bearer",
   "expires_in":3600,
   "scope":"contacts"
}
```

Figure 6: Access Token Response

3. Security Considerations

An audience-restricted access token that is legitimately presented to a resource cannot then be taken by that resource and presented elsewhere for illegitimate access to other resources. The `resource` parameter enables a client to indicate the protected resources where the requested access token will be used, which in turn enables the authorization server to apply the appropriate audience restrictions to the token.

Some servers may host user content or be multi-tenant. In order to avoid attacks where one tenant uses an access token to illegitimately access resources owned by a different tenant, it is important to use a specific resource URI including any portion of the URI that identifies the tenant, such as a path component. This will allow access tokens to be audience-restricted in a way that identifies the tenant and prevents their use, due to an invalid audience, at resources owned by a different tenant.

Although multiple occurrences of the `resource` parameter may be included in a token request, using only a single `resource` parameter is encouraged. If a bearer token has multiple intended recipients (audiences), then the token is valid at more than one protected resource and can be used by any one of those resources to access any of the others. Thus, a high degree of trust between the involved parties is needed when using access tokens with multiple audiences. Furthermore, an authorization server may be unwilling or unable to fulfill a token request with multiple resources.

Whenever feasible, the `resource` parameter should correspond to the network-addressable location of the protected resource. This makes it possible for the client to validate that the resource being requested controls the corresponding network location, reducing the risk of malicious endpoints obtaining tokens meant for other resources. If the `resource` parameter contains an abstract identifier, it is the client's responsibility to validate out of band that any network endpoint to which tokens are sent are the intended audience for that identifier.

4. Privacy Considerations

In typical OAuth deployments the authorization sever is in a position to observe and track a significant amount of user and client behavior. It is largely just inherent to the nature of OAuth, and this document does little to affect that. In some cases, however, such as when access token introspection is not being used, use of the resource parameter defined herein may allow for tracking behavior at a somewhat more granular and specific level than would otherwise be possible in its absence.

5. IANA Considerations

5.1. OAuth Parameters Registration

This specification updates the following value in the IANA "OAuth Parameters" registry [IANA.OAuth.Parameters] established by [RFC6749].

Parameter name: resource
Parameter usage location: authorization request, token request
Change controller: IESG
Specification document(s): RFC 8707

5.2. OAuth Extensions Error Registration

This specification updates the following error in the IANA "OAuth Extensions Error Registry" [IANA.OAuth.Parameters] established by [RFC6749].

Error name: invalid_target
Error usage location: implicit grant error response, token error response
Related protocol extension: resource parameter
Change controller: IESG
Specification document(s): RFC 8707

6. References

6.1. Normative References

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

[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>.

[RFC3986]

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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>.

[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/info/rfc8174>.

6.2. Informative References

[JWT-SAR] Sakimura, N. and J. Bradley, "The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)", Work in Progress, Internet-Draft, draft-ietf-oauth-jwsreq-20, 21 October 2019, <https://tools.ietf.org/html/draft-ietf-oauth-jwsreq-20>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7644] Hunt, P., Ed., Grizzle, K., Ansari, M., Wahlstroem, E., and C. Mortimore, "System for Cross-domain Identity Management: Protocol", RFC 7644, DOI 10.17487/RFC7644, September 2015, <https://www.rfc-editor.org/info/rfc7644>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/info/rfc7662>.

Acknowledgements

This specification was developed within the OAuth Working Group under the chairmanship of Hannes Tschofenig and Rifaat Shekh-Yusef with Eric Rescorla, Benjamin Kaduk, and Roman Danyliw serving as Security Area Directors. Additionally, the following individuals contributed ideas, feedback, and wording that helped shape this specification:

Vittorio Bertocci, Sergey Beryozkin, Roman Danyliw, William Denniss, Vladimir Dzhuvinov, George Fletcher, Dick Hardt, Phil Hunt, Michael Jones, Benjamin Kaduk, Barry Leiba, Torsten Lodderstedt, Anthony Nadalin, Justin Richer, Adam Roach, Nat Sakimura, Rifaat Shekh-Yusef, Filip Skokan, Éric Vyncke, and Hans Zandbelt.

Authors' Addresses

Brian Campbell
Ping Identity
Email: brian.d.campbell@gmail.com

John Bradley
Yubico
Email: ve7jtb@ve7jtb.com

Hannes Tschofenig
Arm Limited
Email: hannes.tschofenig@gmx.net