Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2026

K. McGuinness
Independent
J. Hanson
Keycard Labs

2 March 2026

OAuth 2.0 Resource Parameter in Access Token Response
draft-mcguinness-oauth-resource-token-resp-02

Abstract

This specification defines a new parameter resource to be returned in OAuth 2.0 access token responses. It enables clients to confirm that the issued token is valid for the intended resource. This mitigates ambiguity and certain classes of security vulnerabilities such as resource mix-up attacks, particularly in systems that use the Resource Indicators for OAuth 2.0 specification [RFC8707].

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://mcguinness.github.io/draft-mcguinness-oauth-resource-token-resp/draft-mcguinness-oauth-resource-token-resp.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mcguinness-oauth-resource-token-resp/.

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/mcguinness/draft-mcguinness-resource-token-resp.

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/.

Copyright Notice

Table of Contents

1. Introduction

OAuth 2.0 defines a framework in which clients obtain access tokens from authorization servers and present them to protected resources. In deployments involving multiple protected resources or resource servers (APIs), clients often need to determine whether a given access token is valid for a specific resource.

The Resource Indicators for OAuth 2.0 specification [RFC8707] introduced the resource request parameter, which allows a client to indicate the protected resource(s) for which an access token is intended to be used. An authorization server can use this information when issuing an access token, for example by applying resource-specific restrictions based on policy or configuration.

However, [RFC8707] does not define any mechanism for an authorization server to confirm, in the access token response, which resource(s) were ultimately accepted. As a result, a client has no interoperable way to validate the effective resource scope of an issued access token.

When an authorization request includes one or more resource parameters, authorization servers in deployed systems may exhibit a range of behaviors depending on their capabilities and policy configuration. An authorization server MAY, for example:

* Ignore the resource parameter and issue an access token that is not restricted to any specific resource.

* Ignore the resource parameter and issue an access token that is valid for a server-assigned resource or set of resources.

* Accept all requested resource values and issue an access token that is valid for the complete requested set.

* Accept only a subset of requested resource values and issue an access token that is valid for that subset, without explicitly indicating that other requested resources were rejected.

* Override the requested resource values and issue an access token that is valid for resources determined by authorization server policy or client configuration.

* Reject the request by returning an invalid_target error response as defined in [RFC8707].

In the absence of explicit confirmation in the token response, a client cannot determine which of these behaviors occurred and may incorrectly assume that an access token is valid for a particular resource.

This ambiguity is especially problematic in deployments that rely on dynamic discovery of protected resources and authorization servers. In such environments, a client may learn the protected resource URL at runtime and discover an authorization server using OAuth 2.0 Protected Resource Metadata [RFC9728], without any pre-established trust relationship between the client and the resource. See Section 1.1 for an example of how this can lead to resource mix-up attacks.

A key challenge in these deployments is that the client has no reliable way to validate whether a discovered authorization server is actually authoritative for a given protected resource. While [RFC9728] allows protected resource metadata to be cryptographically signed, this would require clients to be pre-configured with trust anchors for signature verification, which defeats the purpose of runtime discovery.

Similarly, an authorization server could publish a list of protected resources it supports in its metadata [RFC8414], but this approach does not scale in practice for large APIs or resource domains with many distinct resource identifiers, nor does it address cases where authorization server policy dynamically determines resource validity.

Some clients attempt to determine the applicability of an access token by examining audience-related information. A client might call the token introspection endpoint [RFC7662] to obtain an audience value, or might locally inspect the aud claim when the access token is self-contained, such as a JWT [RFC7519]. OAuth treats access tokens as opaque to the client (Section 1.4 of [RFC6749]). As a result, neither token introspection nor local inspection of a self-contained token provides a reliable or interoperable mechanism for determining the protected resource for which an access token is valid.

Audience values are token-format-specific and policy-defined. They do not have standardized semantics across authorization server implementations. As described in Section 2.2, audience values commonly identify authorization servers, tenants, resource servers, or other logical entities rather than concrete protected resource identifiers. A protected resource may accept access tokens with broad, narrow, or indirect audience values. There is no predictable or discoverable relationship between such values and the resource's URL. Clients therefore cannot rely on audience information, whether obtained from a token introspection response or from local inspection of a self-contained access token, to determine whether an access token is applicable to a specific protected resource.

As a result, learning the audience of an issued access token does not provide a reliable or interoperable way for a client to determine whether the token is valid for the intended resource, particularly when multiple protected resources share an authorization server or when the client interacts with resources discovered dynamically at runtime. This document uses the term _resource_ as defined in [RFC8707]. The relationship between resources and token audience values is discussed further in Section 2.2.

Consequently, existing OAuth mechanisms do not provide a practical, interoperable way for a client to confirm that an issued access token is valid for the intended resource.

This specification defines a new resource parameter to be returned in OAuth 2.0 access token responses that is orthogonal to the token format issued by the authorization server. The parameter explicitly identifies the protected resource(s) for which the issued access token is valid, enabling clients to validate token applicability before use and reducing ambiguity across deployments.

1.1. Resource Mix-Up via Dynamic Discovery Example

The following example illustrates how ambiguity about the effective resource scope of an issued access token can lead to a resource mix-up attack in deployments that rely on dynamic discovery.

Preconditions:

* A client wants to access a protected resource at https://api.example.net/data and is not statically configured with knowledge of that resource or its authorization server.

* The client uses OAuth 2.0 Protected Resource Metadata [RFC9728] to dynamically discover an authorization server for the resource.

* An attacker controls the protected resource at https://api.example.net/data and publishes Protected Resource Metadata claiming https://legit-as.example.com as the authorization server, advertising legitimate-looking scopes such as data:read data:write.

* The client has an existing client registration with https://legit-as.example.com.

* The authorization server at https://legit-as.example.com does not support [RFC8707] and ignores the resource parameter, issuing access tokens based solely on requested scopes.

* The user trusts https://legit-as.example.com and would consent to the requested scopes for a legitimate client.

Attack Flow:

1. The client retrieves Protected Resource Metadata from https://api.example.net/data and discovers https://legit-as.example.com as the authorization server.

2. The client sends an authorization request to https://legit-as.example.com, including resource=https://api.example.net/data and scopes data:read data:write.

3. The authorization server processes the request based on scopes, ignores the resource parameter, and—after user consent—issues an access token without confirming the selected resource.

4. The client receives the access token but cannot determine whether it is valid for https://api.example.net/data or for some other protected resource.

5. The client presents the access token to https://api.example.net/data. The attacker intercepts the token and reuses it to access a legitimate protected resource that trusts the same authorization server and accepts tokens with the same scopes.

Without explicit confirmation of the resource in the token response, the client cannot detect that the authorization server ignored or overrode the requested resource indicator. User consent alone may not prevent this attack, particularly when authorization servers do not prominently display resource information during authorization.

2. Conventions and Terminology

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.

2.1. Terminology

The terms "client", "authorization server", "resource server", "access token", "protected resource", "authorization request", "authorization response", "access token request", "access token response" are defined by the OAuth 2.0 Authorization Framework specification [RFC6749].

The term "resource" is defined by the Resource Indicators for OAuth 2.0 specification [RFC8707].

The term "server-assigned resource" denotes resource(s) assigned by the authorization server (e.g., defaults or additions based on policy, configuration, or requested scopes).

The term "StringOrURI" is defined by the JWT specification [RFC7519].

2.2. Resource vs Audience

This specification uses the term _resource_ (as defined in [RFC8707]) rather than _audience_ (as commonly used in access token formats such as JWTs) to describe the protected resource for which an access token is valid. This distinction is intentional and reflects a fundamental separation in OAuth between resource identification and token representation.

In OAuth, a client interacts with a protected resource by making requests to a specific resource URL. The client may not know how that resource is represented internally by the authorization server, including which audience values a resource server will accept. As a result, a client cannot assume any fixed or discoverable relationship between a protected resource's URL and the audience values that may appear in an access token.

While a resource identifier and an access token's audience value may coincide in some deployments, they are not equivalent. A resource server protecting a given resource may accept access tokens with audience restrictions that are:

* *Broad*, such as https://api.example.com, representing an API-wide identifier.

* *Narrow*, such as https://api.example.com/some/protected/resource, representing a specific protected resource.

* *Logical or indirect*, such as urn:example:api or https://example.net, which have no direct correspondence to the resource's URL.

Audience assignment is a matter of authorization server policy. A client therefore cannot reliably predict which audience value will be used in an issued access token or which audience values a resource server will accept. This limitation is particularly relevant in dynamic environments, such as those using OAuth 2.0 Protected Resource Metadata [RFC9728], where a client can discover the protected resource URL but cannot discover the authorization server's audience assignment policy.

Some deployments attempt to encode protected resource identifiers into scope values and rely on the scope parameter in the access token response [RFC6749] to infer the applicability of an access token. This approach conflates two distinct concepts in OAuth. Scope expresses the permissions or actions that an access token authorizes, while the resource parameter [RFC8707] identifies the protected resource at which those permissions may be exercised. Encoding resource identifiers into scope values obscures this distinction, limits the independent evolution of authorization and resource targeting, and does not compose well in deployments where a single resource supports multiple scopes or where the same scope applies across multiple resources.

Some deployments also rely on protected resource metadata [RFC9728] or authorization server metadata [RFC8414] to discover which authorization server is authoritative for a given protected resource. While these mechanisms support discovery, they do not provide issuance-time confirmation of the resource(s) for which an access token is valid. Metadata describes static relationships and capabilities, not the authorization server's resource selection decision for a specific token. As a result, metadata alone cannot be relied upon to determine whether an authorization server honored, restricted, or substituted a requested resource when issuing an access token.

For these reasons, returning audience or scope information in the token response is less useful to a client than returning the resource(s) for which the access token was issued. By returning the resource parameter, this specification enables a client to confirm

that an access token is valid for the specific resource it requested and to detect resource mix-up conditions in which an authorization server issues a token for a different resource than intended.

This approach is consistent with Resource Indicators [RFC8707] and Protected Resource Metadata [RFC9728], which define the resource parameter as the client-facing mechanism for identifying the target protected resource, independent of how a resource server enforces audience restrictions internally.

This specification does not define, constrain, or replace the use of audience values in access tokens, nor does it require any particular token format. How authorization servers encode audience information and how resource servers enforce audience restrictions are explicitly out of scope.

3. Resource Parameter in Token Response

Authorization servers that support this specification MUST include the resource parameter in successful access token responses, as defined in Section 5.1 of [RFC6749], to identify the protected resource(s) for which the access token is valid, according to the rules in Section 3.2.

The value of the resource parameter MUST be either:

* A single case-sensitive string containing an absolute URI, as defined in Section 2 of [RFC8707], when the access token is valid for exactly one resource.

* An array of case-sensitive strings, each containing an absolute URI as defined in Section 2 of [RFC8707], when the access token is valid for more than one resource. The array MUST contain at least one element, and each element MUST be unique when compared using the URI comparison rules in Section 6.2.1 of [RFC3986] after applying syntax-based normalization as defined in Section 6.2.2 of [RFC3986].

When an access token is valid for exactly one resource, the authorization server SHOULD represent the resource parameter value as a single string to improve interoperability and ease of processing. When the access token is valid for more than one resource, the authorization server MUST represent the resource parameter value as an array.

This representation is determined solely by the number of resources for which the access token is valid and applies regardless of how many resource parameters were included in the request.

The resource parameter uses the same value syntax and requirements as the resource request parameter defined in [RFC8707]. Each value MUST be an absolute URI, MUST NOT contain a fragment component, and SHOULD NOT contain a query component.

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

{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "Bearer",
  "expires_in": 3600,
  "resource": "https://api.example.com/"
}
```

3.1. Resource Identifier Comparison

This section defines the canonical rules for comparing resource identifiers when determining uniqueness, evaluating requested resources against authorization server policy, or validating that returned resource values correspond to client expectations.

When comparing resource identifiers, implementations MUST apply the URI comparison rules defined in Section 6.2.1 of [RFC3986], after applying syntax-based normalization as defined in Section 6.2.2 of [RFC3986]. Resource identifiers that are equivalent under these rules MUST be treated as identical.

3.2. Authorization Server Processing Rules

Authorization server processing is determined by the number of resource parameters included in the authorization request or token request, as defined in [RFC8707]. Server-assigned resources may also affect processing; the effective resource set can be influenced by requested scopes, authorization server policy, client configuration, and other factors.

The following rules describe how an authorization server evaluates requested resources and determines the effective resource(s) associated with an issued access token. These rules apply equally to access tokens issued using the authorization code grant and the refresh token grant.

When issuing an access token, any resource parameters included in the token request represent an additional restriction on the resources permitted by the underlying authorization grant. The authorization

server MUST ensure that each requested resource in the token request is within the set of resources authorized by the grant, or otherwise acceptable under local policy consistent with [RFC8707]. If this condition is not met, the authorization server MUST return an invalid_target error and MUST NOT issue an access token.

If the client includes resource parameters in both the authorization request and the token request, the values in the token request MUST be treated as a further restriction and MUST be a subset of the resources permitted by the underlying grant. If no resource parameter is included in the token request, the authorization server MAY issue an access token for the resource(s) previously authorized by the grant, subject to local policy.

When issuing an access token in response to a refresh token request, the authorization server MUST NOT expand the set of authorized resources beyond those previously bound to the underlying grant. If the client supplies one or more resource parameters in the refresh token request, each requested resource MUST be within the previously authorized set or otherwise acceptable under local policy consistent with [RFC8707]. If this condition is not met, the authorization server MUST return an invalid_target error and MUST NOT issue an access token.

An authorization server MAY require clients to include a resource parameter. If a resource parameter is required by local policy and the client does not include one, the authorization server MUST return an invalid_target error as defined in [RFC8707] and MUST NOT issue an access token.

When including a resource parameter in the access token response, the authorization server MUST use a string when exactly one resource value is returned and MUST use an array of strings when more than one resource value is returned (whether from requested resources, server-assigned resources, or both).

When the client included one or more resource parameters in the request, the authorization server MUST NOT issue an access token unless at least one requested resource is acceptable. The server MUST return an invalid_target error if none of the requested resources are acceptable, even if server-assigned resources could be returned. Server-assigned resources may only be included in addition to at least one accepted requested resource.

3.2.1. Summary Table

Client Request Shape	Authorization Server Outcome	Authorization Server Processing Rules
One or more resource values requested	No acceptable requested resource	MUST return invalid_target and MUST NOT issue an access token.
	One or more acceptable requested resources	MUST include resource with at least the accepted requested resource(s). MAY include server-assigned resource(s) in addition.
No resource requested	server-assigned resource(s)	SHOULD include the assigned resource(s) in the resource parameter.
	No resource-specific restriction	SHOULD omit the resource parameter.

Table 1

When comparing resource identifiers, the authorization server MUST apply the rules defined in Section 3.1.

3.2.2. Client Requested One or More Resources

If the client included one or more resource parameters in the authorization request or token request:

* The authorization server MUST evaluate the requested resource value(s) according to local policy and determine which requested values are acceptable.

* If none of the requested resource values are acceptable, the authorization server MUST return an invalid_target error response as defined in [RFC8707] and MUST NOT issue an access token.

* If one or more requested resource values are acceptable:

- The authorization server MUST include the resource parameter in the access token response.

- The returned value(s) MUST include at least one requested resource value according to the rules defined in Section 3.1.

- The returned set MAY contain a strict subset of the requested resource values (when multiple were requested).

- The returned set MAY contain additional server-assigned resources. See Appendix A.2 for an example using scope-defined resources.

- The returned set MUST NOT contain duplicate resource values, including values that differ only by URI normalization using rules defined in Section 3.1.

3.2.3. Client Did Not Request a Resource

If the client did not include any resource parameters in the authorization request or token request:

* If the authorization server assigns one or more server-assigned resource value(s) based on policy, client configuration, or requested scopes:

  - The authorization server SHOULD include the assigned resource value(s) in the resource parameter of the response.

* If the authorization server does not apply any resource-specific restriction to the access token:

  - The authorization server SHOULD omit the resource parameter from the response.

If the resource parameter is omitted, the access token is not valid for any specific resource as defined by this specification.

3.3. Client Processing Rules

A client that supports this extension MUST process access token responses according to the rules in this section, which are determined by whether the client requested one or more resources or no resources, as defined in [RFC8707].

When a resource parameter is included in an access token response, the client MUST interpret and compare the returned resource identifiers using the rules defined in Section 3.1. If the client cannot determine that an access token is valid for the intended protected resource, the client MUST NOT use the access token.

If validation succeeds, the client MAY use the access token and MUST use it only with the resource(s) identified in the response. Any returned scope value MUST be interpreted in conjunction with the returned resource values, and the granted scopes MUST be appropriate for the returned resource(s), consistent with Section 3.3 of [RFC6749].

If validation fails at any point, the client MUST NOT use the access token and SHOULD discard it.

These client processing rules apply equally to access tokens issued using the authorization code grant and to access tokens issued using the refresh token grant.

3.3.1. Summary Table

Client Request Shape	Token Response	Client Processing Rules
One or more resource values requested	resource omitted	Invalid. Client MUST NOT use the access token and SHOULD discard it.
	resource present	Valid only if at least one returned identifier matches a requested resource; additional elements MAY be server-assigned resources.
No resource requested	resource omitted	Valid. Token is not resource-specific.
	resource present	Valid. Client SHOULD treat the returned value as a server-assigned resource assignment.

Any request shape	error=invalid_target	Client MUST treat this as a terminal error and MUST NOT use an access token.

Table 2

3.3.2. Parsing the resource Parameter

If the access token response includes a resource parameter, the client MUST parse it as follows:

* A string value represents a single resource identifier.

* An array value represents multiple resource identifiers; each element MUST be a string.

* Any other value is invalid; the client MUST NOT use the access token and SHOULD discard it.

The client MUST normalize the parsed value to a set of resource identifiers (treating a string as a single-element set) before applying the validation rules below. When comparing resource identifiers, the client MUST apply the rules defined in Section 3.1.

3.3.3. Client Requested One or More Resources

If the client included one or more resource parameters in the token request:

* The response MUST include a resource parameter containing at least one identifier that matches a requested resource (additional identifiers MAY be server-assigned resources).

* Validation fails when the response omits the resource parameter, when the returned set does not contain any of the requested resources, or when duplicate resource identifiers are present.

3.3.3.1. Authorization Request Example

Client obtains an access token for the protected resource at https://api.example.com/customers using the authorization code grant.

3.3.3.1.1. Authorization Request

Client makes an authorization request for the protected resource at https://api.example.com/customers.

```
   GET /authorize?response_type=code
     &client_id=client123
     &redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb
     &scope=customers%3Aread
     &state=abc123
     &resource=https%3A%2F%2Fapi.example.com%2Fcustomers
     &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
     &code_challenge_method=S256
   HTTP/1.1
   Host: authorization-server.example.com
```

3.3.3.1.2. Redirect

The authorization server redirects the user-agent back to the client with an authorization code.

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=abc123
```

3.3.3.1.3. Token Request

The client exchanges the authorization code for an access token.

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

   grant_type=authorization_code&
   code=SplxlOBeZQQYbYS6WxSbIA&
   redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb&
   client_id=client123&
   code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

3.3.3.1.4. Token Response

The authorization server issues an access token that is valid for the protected resource at https://api.example.com/customers.

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

{
  "access_token": "ACCESS_TOKEN",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "customers:read",
  "resource": "https://api.example.com/customers"
}
```

3.3.3.2. Refresh Token Request Example

Client refreshes an access token for the protected resource at https://api.example.com/customers.

3.3.3.2.1. Refresh Token Request

The client uses a refresh token to request a new access token that is valid for the protected resource at https://api.example.com/customers.

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

grant_type=refresh_token&
refresh_token=REFRESH_TOKEN&
client_id=client123&
scope=customers%3Aread&
resource=https%3A%2F%2Fapi.example.com%2Fcustomers
```

3.3.3.2.2. Token Response

The authorization server issues a new access token that is valid for the protected resource at https://api.example.com/customers.

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

{
  "access_token": "ACCESS_TOKEN",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "customers:read",
  "resource": "https://api.example.com/customers"
}
```

3.3.3.3. Authorization Request Example

Client obtains an access token for the protected resources at https://api.example.com/customers and https://api.example.com/orders.

3.3.3.3.1. Authorization Request

Client makes an authorization request for both protected resources.

```
GET /authorize?response_type=code
  &client_id=client123
  &redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb
  &scope=customers%3Aread%20orders%3Aread
  &state=abc123
  &resource=https%3A%2F%2Fapi.example.com%2Fcustomers
  &resource=https%3A%2F%2Fapi.example.com%2Forders
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
  &code_challenge_method=S256
HTTP/1.1
Host: authorization-server.example.com
```

3.3.3.3.2. Redirect

The authorization server redirects the user-agent back to the client with an authorization code.

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=abc123
```

3.3.3.3.3. Token Request

The client exchanges the authorization code for an access token.

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

grant_type=authorization_code&
code=SplxlOBeZQQYbYS6WxSbIA&
redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb&
client_id=client123&
code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

3.3.3.3.4. Token Response

The authorization server issues an access token that is valid for both protected resources.

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

{
  "access_token": "ACCESS_TOKEN",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "customers:read orders:read",
  "resource": [
    "https://api.example.com/customers",
    "https://api.example.com/orders"
  ]
}
```

3.3.3.4. Refresh Token Request Example

Client refreshes an access token for the protected resources at https://api.example.com/customers and https://api.example.com/orders.

3.3.3.4.1. Refresh Token Request

The client uses a refresh token to request a new access token that is valid for both protected resources.

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

grant_type=refresh_token&
refresh_token=REFRESH_TOKEN&
client_id=client123&
scope=customers%3Aread%20orders%3Aread&
resource=https%3A%2F%2Fapi.example.com%2Fcustomers&
resource=https%3A%2F%2Fapi.example.com%2Forders
```

3.3.3.4.2. Token Response

The authorization server issues a new access token that is valid for both protected resources.

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

{
  "access_token": "ACCESS_TOKEN",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "customers:read orders:read",
  "resource": [
    "https://api.example.com/customers",
    "https://api.example.com/orders"
  ]
}
```

3.3.4. Client Did Not Request a Resource

If the client did not include any resource parameters in the token request:

* If the response includes a resource parameter, the client MAY treat it as a server-assigned resource assignment.

* If the response omits the resource parameter, the token SHOULD be treated as unbounded.

3.3.4.1. Authorization Request Example

Client obtains an access token without requesting a specific resource.

3.3.4.1.1. Authorization Request

Client makes an authorization request without including a resource indicator.

```
GET /authorize?response_type=code
  &client_id=client123
  &redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb
  &scope=orders%3Aread
  &state=abc123
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
  &code_challenge_method=S256
HTTP/1.1
Host: authorization-server.example.com
```

3.3.4.1.2. Redirect

The authorization server redirects the user-agent back to the client with an authorization code.

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=abc123
```

3.3.4.1.3. Token Request

The client exchanges the authorization code for an access token.

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

grant_type=authorization_code&
code=SplxlOBeZQQYbYS6WxSbIA&
redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb&
client_id=client123&
code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

3.3.4.1.4. Token Response

The authorization server issues an access token that is valid for the server-assigned protected resource (https://api.example.com/orders).

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

{
  "access_token": "ACCESS_TOKEN",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "orders:read",
  "resource": "https://api.example.com/orders"
}
```

3.3.4.2. Refresh Token Request Example

Client refreshes an access token without explicitly requesting a resource.

3.3.4.2.1. Refresh Token Request

The client uses a refresh token to request a new access token without explicitly requesting a resource.

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

grant_type=refresh_token&
refresh_token=REFRESH_TOKEN&
client_id=client123&
scope=orders%3Aread
```

3.3.4.2.2. Token Response

The authorization server issues a new access token that is valid for the server-assigned protected resource (https://api.example.com/orders).

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

{
  "access_token": "ACCESS_TOKEN",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "orders:read",
  "resource": "https://api.example.com/orders"
}
```

3.3.5. Invalid Resource

An invalid_target error indicates that none of the requested resource values were acceptable to the authorization server. This outcome may result from authorization server policy or client configuration.

Upon receiving an invalid_target error, the client MAY retry the authorization request with a different resource value.

3.3.5.1. Authorization Request Example

Client attempts to obtain an access token for a protected resource (https://unknown.example.com) that is not permitted.

3.3.5.1.1. Authorization Request

Client makes an authorization request for a protected resource that is not permitted.

```
GET /authorize?response_type=code
  &client_id=client123
  &redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb
  &scope=customers%3Aread
  &state=invalid123
  &resource=https%3A%2F%2Funknown.example.com%2F
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
  &code_challenge_method=S256
HTTP/1.1
Host: authorization-server.example.com
```

3.3.5.1.2. Error Redirect

The authorization server rejects the requested resource and does not issue an authorization code.

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=invalid_target&error_description=Resource%2
0not%20allowed&state=invalid123
```

3.3.5.2. Refresh Token Request Example

Client attempts to refresh an access token for a protected resource (https://unknown.example.com) that is not permitted.

3.3.5.2.1. Refresh Token Request

The client uses a refresh token to request a new access token for a protected resource that is not permitted.

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

grant_type=refresh_token&
refresh_token=REFRESH_TOKEN&
client_id=client123&
scope=customers%3Aread&
resource=https%3A%2F%2Funknown.example.com%2F
```

3.3.5.2.2. Error Response

The authorization server rejects the requested resource.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
  "error": "invalid_target",
  "error_description": "Resource not allowed"
}
```

3.4. Security Considerations

This section describes security threats related to ambiguous resource selection and access token reuse, and explains how this specification mitigates those threats, either directly or in combination with other OAuth security mechanisms.

3.4.1. Resource Selection Ambiguity and Mix-Up Attacks

Threat: An authorization server issues a successful access token response without clearly indicating the protected resource(s) for which the token is valid. This can occur when the server applies server-assigned resource selection, narrows a requested resource set, or substitutes a different resource according to local policy without communicating that decision to the client. Such conditions are more likely in deployments where a client interacts with multiple authorization servers and protected resources, or dynamically discovers authorization servers at runtime, including via HTTP authorization challenges using OAuth 2.0 Protected Resource Metadata [RFC9728].

Impact: A client may incorrectly assume that an access token is valid for a different protected resource than the one actually authorized. This can result in access token misuse, unintended token disclosure to an incorrect protected resource, or resource mix-up attacks in which a token intended for one resource is presented to another. These risks are amplified in dynamic environments where the client cannot fully validate the trust relationship between a protected resource and the authorization server.

Mitigation: This specification mitigates resource selection ambiguity and OAuth mix-up attacks by requiring authorization servers to explicitly return the resource(s) for which an access token is valid in the access token response. By providing issuance-time confirmation of the effective resource selection, clients can detect cases where a requested resource was narrowed, substituted, or overridden, and can avoid using access tokens with unintended protected resources.

3.4.2. Limitations of Discovery-Time Mechanisms

Threat: Clients rely on protected resource metadata [RFC9728] or authorization server metadata [RFC8414] to determine which authorization server is authoritative for a protected resource and assume that successful token issuance implies correct resource binding.

Impact: Metadata describes static relationships and supported capabilities, but does not reflect issuance-time authorization decisions. As a result, clients may be unable to detect cases where an authorization server issues an access token valid for a different resource than the one requested, particularly in dynamic or multi-resource environments.

Mitigation: By returning the effective resource selection in the token response, this specification complements discovery-time mechanisms with issuance-time confirmation. This enables clients to verify that an access token is valid for the intended protected resource regardless of how authorization server relationships were discovered.

3.4.3. Token Reuse by Malicious Protected Resources

Threat: A malicious protected resource intercepts an access token during a client's legitimate request and reuses that token to access other protected resources that trust the same authorization server and accept tokens with the same audience and scopes.

Impact: Token reuse can lead to unauthorized access to protected resources beyond those intended by the client, particularly in environments where multiple resources trust a common authorization server and audience values are broad or indirect.

Mitigation: To prevent token reuse attacks, access tokens SHOULD require proof-of-possession, such as Demonstrating Proof-of-Possession (DPoP) as defined in [RFC9449]. Other proof-of-possession mechanisms may also be applicable. Proof-of-possession mechanisms bind the access token to a cryptographic key held by the client and require demonstration of key possession when the token is used. This prevents a malicious protected resource that intercepts an access token from reusing it at other resources, as it does not possess the client's private key. Both the client and authorization server must support proof-of-possession mechanisms for this protection to be effective. See Section 9 of [RFC9449] for additional details.

3.4.4. Client Validation and Defense in Depth

Threat: A client fails to validate the resource(s) returned in the access token response and proceeds to use an access token for a protected resource other than the one for which it was issued.

Impact: Failure to validate the returned resource parameter can result in token misuse or unintended interactions with protected resources, even when the authorization server correctly indicates the effective resource selection.

Mitigation: Clients are advised to validate the resource parameter in the token response as specified in Section 3.3 and to treat mismatches as errors unless explicitly designed to support resource negotiation. While validating the resource parameter provides defense in depth by allowing the client to detect resource substitution, it does not prevent token reuse by malicious protected resources. Clients that require strict resource binding SHOULD treat the absence of the resource parameter as a potential ambiguity.

4. Privacy Considerations

Returning the resource value may reveal some information about the protected resource. If the value is sensitive, the authorization server SHOULD assess whether the resource name can be safely disclosed to the client.

5. IANA Considerations

This document updates the "resource" parameter in the OAuth Parameters registry established by [RFC6749]. The "resource" parameter is defined by [RFC8707] for use in authorization requests and token requests. This specification adds the following usage location:

5.1. OAuth Parameters Registry

Name	Parameter Usage Location	Description	Specification
resource	token response	Resource to which the access token applies	This document

Table 3

6. Normative References

[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>.

[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>.

[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>.

[RFC9728] Jones, M.B., Hunt, P., and A. Parecki, "OAuth 2.0 Protected Resource Metadata", RFC 9728, DOI 10.17487/RFC9728, April 2025, <https://www.rfc-editor.org/info/rfc9728>.

[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>.

[RFC9207] Meyer zu Selhausen, K. and D. Fett, "OAuth 2.0 Authorization Server Issuer Identification", RFC 9207, DOI 10.17487/RFC9207, March 2022, <https://www.rfc-editor.org/info/rfc9207>.

[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>.

[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>.

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

[RFC9700] Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "Best Current Practice for OAuth 2.0 Security", BCP 240, RFC 9700, DOI 10.17487/RFC9700, January 2025, <https://www.rfc-editor.org/info/rfc9700>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/info/rfc9449>.

[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>.

[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>.

Appendix A. Additional Examples

A.1. Requesting a token for a dynamically discovered protected resource

The following example details the need for the resource parameter when a client dynamically discovers an authorization server and obtains an access token using [RFC9728] and [RFC8414]

Client attempts to access a protected resource without a valid access token

```
GET /resource
Host: api.example.com
Accept: application/json
```

Client is challenged for authentication

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer resource_metadata=
  "https://api.example.com/.well-known/oauth-protected-resource"
```

Client fetches the resource’s OAuth 2.0 Protected Resource Metadata per [RFC9728] to dynamically discover an authorization server that can issue an access token for the resource.

```
GET /.well-known/oauth-protected-resource
Host: api.example.com
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json

{
   "resource":
     "https://api.example.com/resource",
   "authorization_servers":
     [ "https://authorization-server.example.com/" ],
   "bearer_methods_supported":
     ["header", "body"],
   "scopes_supported":
     ["resource.read", "resource.write"],
   "resource_documentation":
     "https://api.example.com/resource_documentation.html"
 }
```

Client discovers the Authorization Server configuration per [RFC8414]

```
GET /.well-known/oauth-authorization-server
Host: authorization-server.example.com
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json

{
  "issuer": "https://authorization-server.example.com/",
  "authorization_endpoint": "https://authorization-server.example.com/oauth2/authorize",
  "token_endpoint": "https://authorization-server.example.com/oauth2/token",
  "jwks_uri": "https://authorization-server.example.com/oauth2/keys",
  "scopes_supported": [
    "resource.read", "resource.write"
  ],
  "response_types_supported": [
    "code"
  ],
  "grant_types_supported": [
    "authorization_code", "refresh_token"
  ],
  ...
}
```

Client makes an authorization request for the resource

```
GET /oauth2/authorize?response_type=code
  &client_id=client123
  &redirect_uri=https%3A%2F%2Fclient.example%2Fcallback
  &scope=resource.read
  &state=abc123
  &resource=https%3A%2F%2Fapi.example.com%2Fresource
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
  &code_challenge_method=S256
HTTP/1.1
Host: authorization-server.example.com
```

User successfully authenticates and delegates access to the client for the requested resource and scopes

```
HTTP/1.1 302 Found
Location: https://client.example/callback?code=SplxlOBeZQQYbYS6WxSbIA&state=abc123
```

Client exchanges the authorization code for an access token

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

grant_type=authorization_code&
code=SplxlOBeZQQYbYS6WxSbIA&
redirect_uri=https%3A%2F%2Fclient.example%2Fcallback&
client_id=client123&
code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

Client obtains an access token for the resource.

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

{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "resource.read",
  "resource": "https://api.example.com/resource"
}
```

Client verifies that it obtained an access token that is valid for the discovered resource.

A.2. Scope-defined resource with a resource request

An authorization server may determine resources from requested scopes in addition to explicit resource parameters. For example, a scope might imply a protected resource URL that the authorization server returns as a server-assigned resource alongside any client-requested resources.

The following example uses OpenID Connect: the openid scope implies access to the UserInfo endpoint. When a client requests both a protected resource and the openid scope, the authorization server may determine the UserInfo endpoint URL from its configuration (e.g., {issuer}/userinfo) and return it as a server-assigned resource alongside the client-requested resource.

A.2.1. Authorization request example

A.2.1.1. Authorization Request

```
   GET /authorize?response_type=code
     &client_id=client123
     &redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb
     &scope=openid%20profile%20data%3Aread
     &state=oidc123
     &resource=https%3A%2F%2Fapi.example.com%2Fdata
     &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
     &code_challenge_method=S256
   HTTP/1.1
   Host: idp.example.com
```

A.2.1.2. Redirect

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=oidc123
```

A.2.1.3. Token Request

```
   POST /token HTTP/1.1
   Host: idp.example.com
   Content-Type: application/x-www-form-urlencoded

   grant_type=authorization_code&
   code=SplxlOBeZQQYbYS6WxSbIA&
   redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb&
   client_id=client123&
   code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

A.2.1.4. Token Response

The authorization server issues an access token valid for both the requested resource (https://api.example.com/data) and the UserInfo endpoint (https://idp.example.com/userinfo), which it determined from the openid scope.

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

{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "openid profile data:read",
  "resource": [
    "https://api.example.com/data",
    "https://idp.example.com/userinfo"
  ]
}
```

The client validates that the requested resource (https://api.example.com/data) is present in the response. The additional resource (https://idp.example.com/userinfo) is a server-assigned resource determined by the authorization server based on the openid scope, enabling the client to use the same access token for both the API and the UserInfo endpoint.

Acknowledgments

This proposal builds on prior work in OAuth 2.0 extensibility and security analysis, particularly [RFC8707], [RFC9700], and [RFC9207].

The authors would like to thank the following individuals who contributed ideas, feedback, and wording that helped shape this specification: Filip Skokan

Document History

-02

* Modified processing rules to allow server-assigned resources to be combined with requested resources

* Added OIDC example

* Collapsed one and many processing rules to single "one or more"

* Changed OAuth Registry to update existing resource parameter

* Editorial cleanup and consistency updates

* Updated Acknowledgements

-01

* Revised Introduction and included attack example

* Added Resource vs Audience to Terminology

* Revised Response to provide detailed Authorization Server and Client Processing Rules

* Updated Security Considerations

* Editorial cleanup and consistency

-00

* Initial revision

Authors' Addresses

Karl McGuinness
Independent
Email: public@karlmcguinness.com

Jared Hanson
Keycard Labs
Email: jared@keycard.ai
URI: https://keycard.ai