Abstract

This document specifies a new parameter "authorization_details" that is used to carry fine grained authorization data into the OAuth authorization request.
1. Introduction

[RFC6749] defines the parameter "scope" that allows OAuth clients to specify the expected scope, i.e. the permission, of an access token. This mechanism is sufficient to implement static scenarios and course grain authorization requests, such as "give me read access to the resource owner’s profile" but it’s not sufficient to specify fine grained authorization requirements, such as "please let me make a payment with the amount of 45 Euros" or "please give me read access to folder A and write access to file X".

This draft introduces a new parameter "authorization_details" that allows clients to specify their fine grained authorization requirements using the expressiveness of JSON data structures.

For example, a request for payment authorization can use a JSON object like this:
{  
  "instructedAmount":{  
    "currency":"EUR",  
    "amount":"123.50"  
  },  
  "debtorAccount":{  
    "iban":"DE40100100103307118608"  
  },  
  "creditorName":"Merchant123",  
  "creditorAccount":{  
    "iban":"DE02100100109307118603"  
  },  
  "remittanceInformationUnstructured":"Ref Number Merchant"  
}

For a comprehensive discussion of the challenges arising from new use cases in the open banking and electronic signing spaces see [transaction-authorization].

1.1. 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.

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. Request parameter "authorization_details"

The request parameter "authorization_details" contains a JSON object. This JSON object is composed of one or more JSON objects on the second level, each of them containing the data to specify the authorization requirements for a certain type of resource. The type of resource or access requirement is determined by the name of the JSON object.

This example shows the specification of authorization details for a payment initiation transaction:
This example shows a combined request asking for access to account information and allowance to initiate a payment:
The named JSON objects "account" and "payment" represent the different authorization data to be used by the AS to ask for consent and MUST subsequently also be made available to the respective RSs.

2.1. Authorization data elements types

It is assumed that the structure of each of the authorization data elements is tailored to the needs of a certain application, API, or resource type. For example, the example structures shown above are based on certain kinds of APIs that can be found in the Open Banking space.

This draft therefore only defines the "authorization_details" element (the container) and a minimal set of requirements regarding the structure of the contained authorization data elements.

Note: different applications MUST ensure that their authorization data types do not collide. This is either achieved by using a
namespace under the control of the entity defining the type name or by registering the type with the new "OAuth Authorization Data Type Registry" (see Section 9).

The following example shows how an implementation could utilize the namespace "https://scheme.examples.org/" to ensure collision resistant element names.

```json
{
  "https://scheme.examples.org/files": {
    "permissions": [
      {
        "path": "/myfiles/A",
        "access": [
          "read"
        ]
      }
    ]
  }
}
```

2.2. Multiple instances of the same authorization data type

It's possible that the client asks for different kinds of access to different resources of the same type. There are two ways to cope with such a situation.

For some applications, e.g. file access, it might be reasonable to build a way to allocated authorization data to certain resources into the application specific JSON structure.

Here is an example:
{  
  "https://scheme.examples.org/files":{  
    "permissions":{  
      {  
        "path":"/myfiles/A",
        "access":{  
          "read"  
        }  
      },  
      {  
        "path":"/myfiles/A/X",
        "access":{  
          "read",
          "write"  
        }  
      }  
    }  
  }  
}

Alternatively, a client MAY specify multiple instances of the same authorization data element type and distinguish those elements by adding a suffix "$"+"<instancename>". The following shows authorization details for requesting access to different folder on different IMAP servers (assuming the resource owner has access to both of them):

{  
  "https://scheme.examples.org/imap":{  
    "server":"imap.example.com",
    "mailbox":"/users/<current>",
    "access":{  
      "read",
      "write"  
    }  
  },  
  "https://scheme.examples.org/imap$2":{  
    "server":"imap.example.org",
    "mailbox":"/users/shared/folder3",
    "access":{  
      "read"  
    }  
  }  
}
2.3. Using "authorization_details"

The request parameter can be used anywhere where the "scope" parameter is used, examples include:

- Authorization requests as specified in [RFC6749],
- Request objects as specified in [I-D.ietf-oauth-jwsreq],
- Device Authorization Request as specified in [RFC8628]

Parameter encoding is determined by the respective context.

In the context of an authorization request according to [RFC6749], the parameter is encoded using the "application/x-www-form-urlencoded" format as shown in the following example (JSON string trimmed for brevity):

```
GET /authorize?response_type=code&client_id=s6BhdRkt3
    &state=af0ifjsldkJ
    &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
    &code_challenge_method=S256,
    &code_challenge=5c305578f8f19b2dcd6c3c955c0a...97e43917cd,
    &authorization_details=%7B%22payment%22%3A%7B%22instructedAmount
        %22%3A%7B%22currency%22%3A%22EUR%22%2C%22amount%22%3A%22123.50%2
        2%7D%2C%22debtorAccount%22%3A%7B%22iban%22%3A%22DE40100100103307
        118608%22%7D%2C%22creditorName%22%3A%22Merchant123%22%2C%22creditori
        torAccount%22%3A%7B%22iban%22%3A%22DE02100100109307118603%22%7D%2
        C%22remittanceInformationUnstructured%22%3A%22Ref%20Number%20Merchant%22%7D%2C%22%7D%2C%22%7D%2C%22%7D HTTP/1.1
Host: server.example.com
```

In the context of a request object, "authorization_details" is added as another top level JSON element.


```json
{
   "iss": "s6BhdRkqt3",
   "aud": "https://server.example.com",
   "response_type": "code",
   "client_id": "s6BhdRkqt3",
   "redirect_uri": "https://client.example.com/cb",
   "state": "af0ifjsldkj",
   "code_challenge_method": "S256",
   "code_challenge": "5c305578f8f19b2dcd6c3c955c0a...97e43917cd",
   "authorization_details": {
      "payment": {
         "instructedAmount": {
            "currency": "EUR",
            "amount": "123.50"
         },
         "debtorAccount": {
            "iban": "DE40100100103307118608"
         },
         "creditorName": "Merchant123",
         "creditorAccount": {
            "iban": "DE02100100109307118603"
         },
         "remittanceInformationUnstructured": "Ref Number Merchant"
      }
   }
}
```

Note: Authorization request URIs containing authorization details in a request parameter or a request object can become very long. Implementers SHOULD therefore consider to use the "request_uri" parameter as defined in [I-D.ietf-oauth-jwsreq], potentially in combination with the pushed request object mechanism as defined in [PRO] to pass authorization details in a reliable and secure manner.

2.4. Authorization Request Processing

Based on the data provided in the "authorization_details" parameter the AS will ask the user for consent to the requested access permissions.

Note: the AS is supposed to merge the authorization requirements given in the "scope" parameter and the "authorization_details" parameter if both are present in the authorization request.

The AS MUST refuse to process any unknown authorization data type. If the "authorization_details" contains any unknown authorization data type, the AS MUST abort processing and respond with an error "invalid_scope" to the client.
If the resource owner grants the client the requested access, the AS will issue tokens to the client that are associated with the respective "authorization_details".

The AS MUST make the "authorization_details" available to the respective resource servers. The AS MAY add the "authorization_details" element to access tokens in JWT format and to Token Introspection responses.

The AS MUST take into consideration the privacy implications when sharing authorization details with the resource servers. The AS SHOULD share this data with the resource servers on a "need to know" basis.

2.5. Token Response

In addition to the token response parameters as defined in [RFC6749], the authorization server MUST also return the authorization details as granted by the resource owner and assigned to the respective access token.

This is shown in the following example:
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
   "access_token":"2YotnF2FEjr1zCsicMWpAA",
   "token_type":"example",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
   "authorization_details":{
      "payment":{
         "instructedAmount":{
            "currency":"EUR",
            "amount":"123.50"
         },
         "debtorAccount":{
            "iban":"DE40100100103307118608"
         },
         "creditorName":"Merchant123",
         "creditorAccount":{
            "iban":"DE02100100109307118603"
         },
         "remittanceInformationUnstructured":
            "Ref Number Merchant"
      }
   }
}

2.6. Relationship to "resource" parameter

[I-D.ietf-oauth-resource-indicators] defines the request parameter
"resource" indicating to the AS the resource(s) where the client
intends to use the access tokens issued based on a certain grant.

This mechanism is a way to audience restrict access tokens and to
allow the AS to create resource specific access tokens.

This draft can be used in conjunction with
[I-D.ietf-oauth-resource-indicators] in the same way as the "scope"
parameter. The AS is supposed to narrow down the authorization
details and respective permissions to the needs of the particular
resource when minting an access token.

While this depends on the AS to know what authorization details are
relevant for what RS, this draft can also be combined with the
concept of resource indicators to make this relationship explicit
and to narrow the privileges of an access token down to certain
permissions given on a certain resource down to the individual operation (see [I-D.ietf-oauth-security-topics], section-3.3).

As an example, it is possible to specify that the client will get "read" access to "file X" stored at the resource "https://store.example.com" [1]. To achieve this, the example given above for access to an IMAP server is slightly modified to use the "resource" element as part of the top level claims within the authorization data element.

```
{
  "https://scheme.examples.org/imap":{
    "resource":"imap.example.com",
    "mailbox":"/users/<current>",
    "access":[
      "read",
      "write"
    ]
  },
  "https://scheme.examples.org/imap$2":{
    "resource":"imap.example.org",
    "mailbox":"/users/shared/folder3",
    "access":[
      "read"
    ]
  }
}
```

The AS MUST respect the value of the "resource" element when deciding whether a certain element is placed into a (structured) access token or token introspection response.

3. Metadata

TBD

The AS advertises support for "authorization_details" using the metadata parameter "authorization_details_supported" of type boolean.

The authorization data types supported can be determined using the metadata parameter "authorization_data_types_supported", which is an JSON array.

Clients announce the authorization data types the use in the new dynamic client registration parameter "authorization_data_types".

The registration of new authorization data types with the AS is out of scope of this draft.
4. Further Examples

TBD

- self contained (account information, claims, signing)
- external reference (payment)
- multiple payments
- access to e-mail
- access to files/directories

5. Implementation Considerations

The scheme and processing will significantly vary among different authorization data types. Any implementation of this draft is therefore supposed to allow the customization of the user consent and the handling of access token data.

One option would be to have a mechanism allowing the registration of extension modules, each of them responsible for rendering the respective user consent and any transformation needed to provide the data needed to the resource server by way of structured access tokens or token introspection responses.

6. Security Considerations

Authorization details are sent through the user agent in case of an OAuth authorization request, which makes them vulnerable to modifications by the user. In order to ensure their integrity, the client SHOULD send authorization details in a signed request object as defined in [I-D.ietf-oauth-jwsreq] or use the "request_uri" authorization request parameter as defined in [I-D.ietf-oauth-jwsreq] to pass the URI of the request object to the authorization server.

7. Privacy Considerations

Implementers MUST design and use authorization details in a privacy preserving manner. Any sensitive personal data included in authorization details MUST be prevented from leakage, e.g. through referrer headers. Implementation options include encrypted request objects as defined in [I-D.ietf-oauth-jwsreq] or transmission of authorization details via end-to-end encrypted connections between client ans authorization server by utilizing the "request_uri" authorization request parameter as defined in [I-D.ietf-oauth-jwsreq].
8. Acknowledgements

I would like to thank Brian Campbell, Daniel Fett, Sebastian Ebling, Dave Tonge, Mike Jones, Nat Sakimura, Rob Otto, and Justin Richer for their valuable feedback during the preparation of this draft.

I would like to thank Dave Tonge and Aaron Parecki for their valuable feedback.

9. IANA Considerations

- "authorization_details" as JWT claim
- "authorization_details_supported" and "authorization_data_types_supported" as metadata parameters
- "authorization_data_types" as dynamic client registration parameter
- establish authorization data type registry

10. References

10.1. Normative References


10.2. Informative References

[I-D.ietf-oauth-jwsreq]

[I-D.ietf-oauth-resource-indicators]

[I-D.ietf-oauth-security-topics]

[PRO]

[transaction-authorization]

10.3. URIs

[1] https://store.example.com"

Appendix A. Document History

[[ To be removed from the final specification ]]

-01

o Added Security Considerations

o Added Privacy Considerations

o Added notes on URI size and authorization details

o Added requirement to return the effective authorization details granted by the resource owner in the token response

-00