Rich Authorization Requests

draft-ietf-oauth-rar

05.04.2021

Brian Campbell, Justin Richer, Torsten Lodderstedt

Rich Authorization Requests

new parameter "authorization_details" allows to convey fine grained and structured authorization data as JSON objects

designed to be used where “scope” is not sufficient

Inspired by use cases and solutions in:

- Open Banking
- eHealths
- eSigning
- eGovernment

```
[
 {
  "type": "payment_initiation",
  "instructedAmount": {
    "currency": "EUR",
    "amount": "123.50"
  },
  "creditorName": "Merchant",
  "creditorAccount": {
    "iban": "DE021001...7118603"
  }
]
```

Changes since IETF-107

- 3 new revisions
- Restructured draft for better readability
- Clarifications
  - dependencies between "resource" and "authorization_details" parameters
  - authorization details enrichment
  - unknown authorization details parameters
- Added implementation considerations
- (Continuous) synchronization with GNAP

Implementation Considerations

- Processing and presentation of authorization details will vary significantly among different authorization data types.
- Products should allow deployments
  - to determine presentation of the authorization_details
  - modification of requested authorization_details in the user consent process, e.g. adding fields
  - allow merge of requested and pre-existing authorization_details
- Design options (non-exhaustive)
  - Redirect from product to custom module
  - Callback from product to custom module
  - Custom module built on top of product API
  - Custom build (e.g. fork of open source project)

Open Topic: authorization_details token request parameter

- Assign privileges to first access token (code)
- Downscope privileges of pre-existing grant (code, refresh token, CIBA, device)
- Request access tokens with client credentials

Requested and granted authorization details need to be compared

Comparing Authorization Details

Comparing Scopes

- What's supposed to happen:
  - `"a b c"` is requesting more than `"a b"`
- What sometimes happens:
  - `"c"` is included in the request for `"a"`
  - `"b"` turns on some special functionality instead of asking for access at an RS
- Real-world examples:
  - GitHub API `"repo"` vs `"repo:status"`
  - OpenID Connect `"openid"` and `"offline_access"`
- Still possible to do a simple set comparison and mostly get away with it

Comparing authorization details

- Don’t say anything?
  - Hope for the best!
- Compare JSON objects?
  - Normalization required
  - Makes assumptions about API design
- Leave it out of scope
  - Fully defined by `type` value
- Editors’ proposal:
  - Give some examples for comparison practices, but leave it up to the `type` definition

Comparing two requests: the simple case

```
{
  "type": "photo-api",
  "actions": [
    "read"
  ],
  "locations": [
    "https://server.example.net/"
  ],
  "datatypes": [
    "images"
  ]
}
```

```
{
  "type": "photo-api",
  "actions": [
    "read", "write"
  ],
  "locations": [
    "https://server.example.net/",
    "https://resource.local/other"
  ],
  "datatypes": [
    "metadata", "images"
  ]
}
```

Compare object members: more values == more access

Comparing two requests: subsumption

```
{
  "type": "photo-api",
  "actions": [
    "read"
  ],
  "locations": [
    "https://server.example.net/"
  ],
  "datatypes": [
    "images"
  ]
}
```

```
{
  "type": "photo-api",
  "actions": [
    "write"
  ],
  "locations": [
    "https://example.net/"
  ],
  "datatypes": [
    "metadata"
  ]
}
```

Compare object members: some values subsume others

Comparing two requests: defaults

```
{
  "type": "photo-api",
  "actions": [
    "read"
  ],
  "locations": [
    "https://server.example.net/"
  ],
  "datatypes": [
    "images"
  ]
}
```

```
{
  "type": "photo-api",
  "actions": [
    "read"
  ]
}
```

Compare object members: AS has defaults for some items

Comparing two requests: added detail

```
{
  "type": "photo-api",
  "actions": [
    "read"
  ],
  "locations": [
    "https://server.example.net/"
  ],
  "datatypes": [
    "images"
  ]
}
```

```
{
  "type": "photo-api",
  "actions": [
    "read"
  ],
  "locations": [
    "https://server.example.net/"
  ],
  "datatypes": [
    "images"
  ],
  "identifier": "S2B-7C2-MY2Y"
}
```

Compare object members: add more specific detail with new field

Comparing two requests: more objects

```
[
  {
    "type": "photo-api",
    "actions": [
      "write"
    ],
    "datatypes": [
      "images"
    ]
  }
]
```

```
[
  {
    "type": "photo-api",
    "actions": [
      "write"
    ],
    "datatypes": [
      "images"
    ]
  },
  {
    "type": "photo-api",
    "actions": [
      "read"
    ],
    "datatypes": [
      "metadata"
    ]
  }
]
```

Compare arrays: how does a request match across objects?

Comparing two requests: arbitrary API designs

```
{
  "type": "arbitrary-api",
  "foo": [
    "bar"
  ],
  "baz": true
}
```

```
{
  "type": "arbitrary-api",
  "foo": [
    "batman"
  ],
  "quux": "quuuuuuuux"
}
```

Compare object members: BUT HOW??

Which is correct?

- All of them
  - Depends on the nature of the API being protected and described
  - OAuth doesn’t take a stance on the nature of the API

Provide guidance

- Concepts of a request being “more” or “less” than another
  - Needed in refresh tokens, user consent, authorization
- API designers need to consider this when defining the `type` they use
- AS implementers need to make comparisons
  - Custom: whatever makes sense for the API
  - General-purpose: pluggable comparison system? (see implementation considerations)
- Spec can show common patterns as examples