Rich Authorization Requests

BOF Transactional OAuth, 18.11.2019
Torsten Lodderstedt, yes.com
Focus

- Use of OAuth in security sensitive scenarios, like
  - Open Banking
  - Strong Identity Attestation
  - (Qualified) Electronic Signatures
  - eHealth
  - eGovernment
Informed by

- Work on establishing an open banking ecosystem at yes.com
- Support of open banking API initiatives (PSD2 context)
- Work at OpenID Foundation’s Financial-Grade API WG
- Work at Cloud Signature Consortium
- Discussions with people who work in eHealth and eGovernment
Example: Authorization in Financial APIs
Requirements from PSD2 regulation

- **Consent**: customer consent is required, either for
  - individual requests or
  - as mandate for designated payment accounts and associated payment transactions

- **Dynamic Linking**: payment initiation requests must be bound to amount and payee as approved by the customer
Example Authorization Data

```
{
    "instructedAmount":{
        "currency":"EUR",
        "amount":"123.50"
    },
    "debtorAccount":{
        "iban":"DE40100100103307118608"
    },
    "creditorName":"Merchant123",
    "creditorAccount":{
        "iban":"DE02100100109307118603"
    },
    "remittanceInformationUnstructured":"Ref Number Merchant"
}
```

User needs to consent to and RS needs to enforce this scope!
Example: Access to Account Information

```
{
    "access":{
        "balances":[
            {
                "iban":"DE40100100103307118608"
            },
            {
                "iban":"DE67100100101306118605"
            }
        ],
        "transactions":[
            {
                "iban":"DE40100100103307118608"
            }
        ],
        "validUntil":"2017-11-01"
    }
}
```

List of accounts and respective permissions + duration of the grant
Qualified Electronic Signature

{
    "credentialID":"60916d31-932e-4820-ba82-1fceed1c9ea3",
    "documentDigests": [ 
        {
            "hash":"sTOgwOm+474gfj0q0x1iSNspKqbcse4Ieiq1Dg/HWUI=",
            "label":"Credit Contract"
        },
        {
            "hash":"HZQzZmMAIEkfGH0/ZKWihsd0xg3H6bZYztgsMTLw0=",
            "label":"Contract Payment Protection Insurance"
        }
    ],
    "hashAlgorithmOID":"2.16.840.1.101.3.4.2.1"
}
Example: OpenID Connect

```json
{
  "userinfo":{
    "email":{
      "essential":true
    },
    "email_verified":{
      "essential":true
    },
    "given_name":null,
    "family_name":{
      "value":"Meier"
    },
    "birthdate":null,
    "place_of_birth":null,
    "nationality":null,
    "address":null
  }
}
```

Privacy by Design & Data Minimization require RP to list individual claims
Commonalities

- Privileges very narrowly defined (and must also be enforced)
- Authorization data fine grained & structured (voluminous)
- Transaction authorization (one time & transaction specific values)
- Authorization data may contain PII - confidentiality is important
- Integrity and authenticity is generally a key requirement
Problem Statement **Transport**

- OAuth authorization code flow sends parameters as URI query parameters via redirection in the user-agent
- **Challenges**
  - There is no cryptographical integrity and authenticity protection
  - There is no mechanism to ensure confidentiality of the request parameters.
  - Authorization request URLs can become quite large in the scenarios just described.
Problem Statement **Representation**

- Expressiveness of scopes is not sufficient for the scenarios just explained
  - No structure, no dynamic values - made for simple static access requests
  - Ambiguous ("openid email read")
- Allocation of requested permissions to resource server specific access tokens is hard (despite resource indicators)
(Selected) Solutions from the Wild
Additional Parameter

- **OpenID Connect:**
  - **claims** parameter
  - (signed) request objects
- **PolishAPI:**
  - **scope_details** parameter in authorization request as HTTP POST
  - Transaction specific Redirect URI for authorization

```json
POST https://api.testbank.com/v1/payments/sepa-credit-transfers
Content-Type: application/json
...
{
  ..
  "scope": "pis",
  "scope_details": {
    "privilegeList": [
      {
        "pis:domestic": {
          "domesticPaymentRequest": {
            "recipient": {
              "accountNumber": "PL85114000000000000000000100",
              "name": "Jan Kowalski"
            },
            "sender": {},
            "transferData": {
              "description": "Transfer",
              "amount": "0.01",
              "currency": "PLN"
            },
            "deliveryMode": "STANDARD_D1"
          }
        }
      }
    ],
    "scopeGroupType": "PIS",
    "consentId": "b72fca1d-a2d6-486f-8f98-32f81459ad6f",
    "scopeTimeDuration": 5,
    "throttlingPolicy": "PSD2_REGULATORY"
  }
}```
Lodging Intent

- UK OB, NextGenPSD2, yes.com
- External resource contains the authorization details
- Authorization request refers to external resource (claims field or dynamic scope value part)

```
POST /payments HTTP/1.1
Host: api.bank.example
Content-Type: application/json
Authorization: Bearer eyJraWQiOiJOQnlW...

{
    "creditor": "DE56378485858575858585",
    "instructedAmount": {
        "currency": "EUR",
        "amount": "123"
    },
    "remittanceInformationUnstructured": "Ref Number Merchant: ..."
}
```

HTTP/1.1 201 Created
Content-Type: application/json
Location: /payments/36fc67776

```
{
    "consentId": "36fc67776"
}
```

```
GET /authorise?response_type=code&
client_id=3630BF72-E979-477A-A8FF-8A338F07C852&
redirect_uri=https%3A%2F%2Fclient%2Eexample%2Fcb&
scope=payment%3A36fc67776&
state=S8NJ7uqk5fY4EjNvP_G_FtyJu6pUsvH9jsYni9dMAJw&
code_challenge_method=S256&
code_challenge=5c305578f8f19b2dcd8c3c955c0aa...
43917cd0f36 HTTP/1.1
Host: as.bank.example
```
Generic OAuth Solution?
Pushed Authorization Requests

Pushed Authorization Requests (Overview)

- Based on previous work at OpenID Foundation’s FAPI working group
- Draft authors: Brian Campbell, Nat Sakimura, Dave Tonge, Filip Skokan, Torsten Lodderstedt
- PAR complements JAR by providing an interoperable way to push the payload of an authorization request object directly to the AS in exchange for a "request_uri".
- Provided via new **pushed authorization request endpoint**
How does it look like?
Traditional OAuth Authorization Request

GET /authorize?response_type=code
&client_id=s6BhdRkqt3
&state=af0ifjsldkj
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1
    Host: as.example.com
Pushed Authorization Request

POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnlxS3REUmJuZIzkbaUl3
response_type=code&
client_id=s6BhdRkqt3&
state=af0ifjsldkj&
redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
Pushed Authorization Response

HTTP/1.1 201 Created
Cache-Control: no-cache, no-store
Content-Type: application/json

{
    "request_uri": "urn:example:bwc4JK-ESC0w8acc191e-Y1LTC2",
    "expires_in": 90
}
Authorization Request (according to JAR)

GET /authorize?request_uri=
    urn%3Aexample%3Abwc4JK-ESC0w8acc191e-Y1LTC2 HTTP/1.1
Pushed Request Object

POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZIZkbUI3

request=eyJraWQiOiJrMmJkYyIsImFsZyI6Ii1lJTMjU2In0.eyJpc3MiOiJzNkJoZFFJrcXQzliwiYXVkljoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJyZXNwb25zZV90eXBlljoiY29kZSIiLCJhZjBpZmpzbGRraIiwiYXVkIjoiaHR0cDovL2NyZWF0aW9ucy5jb20iLCJyZWRpcmVjdF91cmkiOiJodHRwczovL2RlZC5jb20vd3d3LmNvbS8iLCJyZWRpcmVjdF91cGxvdyI6IiJ9

NEW

POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZIZkbUI3

request=eyJraWQiOiJrMmJkYyIsImFsZyI6Ii1lJTMjU2In0.eyJpc3MiOiJzNkJoZFFJrcXQzliwiYXVkljoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJyZXNwb25zZV90eXBlljoiY29kZSIiLCJhZjBpZmpzbGRraIiwiYXVkIjoiaHR0cDovL2NyZWF0aW9ucy5jb20iLCJyZWRpcmVjdF91cmkiOiJodHRwczovL2RlZC5jb20vd3d3LmNvbS8iLCJyZWRpcmVjdF91cGxvdyI6IiJ9
Pushed Authorization Response

HTTP/1.1 201 Created
Cache-Control: no-cache, no-store
Content-Type: application/json

{
    "request_uri": "urn:example:bwc4JK-ESC0w8acc191e-Y2LTC2",
    "expires_in": 90
}
Authorization Request (according to JAR)

GET /authorize?request_uri=
urn%3Aexample%3Abwc4JK-ESC0w8acc191e-Y2LTC2 HTTP/1.1
Advantages

- Significantly improved security ...
  - Request Integrity
  - Client authentication
- ... and robustness ...
- ... while offering a simple migration path
- Higher security level by passing signed/encrypted request objects
- redirect_uri can be dynamically registered for confidential clients
- Seems to be resistant against mix-up (analysis ongoing)
Rich Authorization Requests

Rich Authorization Requests

- Based on work in the FAPI WG and on OAuth.xyz
- Authors: Justin Richer, Brian Campbell, Torsten Lodderstedt
- Introduces new parameter `authorization_details` that is used to carry fine grained authorization data in the OAuth authorization request as typed JSON objects
- Can be used in addition or instead of the `scope` parameter in OAuth 2.0
- Same data structure is used in OAuth.xyz
Authorization_details (Syntax)

```
[
  {
    "type":"payment_initiation",
    "actions":[
      "initiate",
      "status",
      "cancel"
    ],
    "locations":{
      "https://example.com/payments"
    },
    "instructedAmount":{
      "currency":"EUR",
      "amount":"123.50"
    },
    "debtorAccount":{
      "iban":"DE40100100103307118608"
    },
    "creditorName":"Merchant123",
    "creditorAccount":{
      "iban":"DE02100100109307118603"
    },
    "remittanceInformationUnstructured":"Ref Number Merchant"
  }
]
```

- Array of JSON objects, each of them specifying a set of permissions a client wants to obtain
- Element structure determined by type field
- locations should be used to assign every element to a resource server (audience)
- draft introduces further common data types, e.g. actions
authorization_details (Usage Examples)

**URI parameter**

GET /authorize?response_type=code
   &client_id=s6BhdRkqt3
   &state=af0ifjsldkj
   &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcba
   &code_challenge_method=S256
   &code_challenge=K2-ltc83acc4h0c9w6ESC_rEMTJ3bww-uCHaoeK1t8U

```
{  
   "iss": "s6BhdRkqt3",  
   "aud": "https://server.example.com",  
   "response_type": "code",  
   "...
   "authorization_details": [  
      {  
         "type": "payment_initiation",  
         "actions": [  
            "initiate",  
            "status",  
            "cancel"  
         ],  
         "locations": [  
            "https://example.com/payments"  
         ],  
         "instructedAmount": {  
            "currency": "EUR",  
            "amount": "123.50"  
         }  
      }  
   ]
}
```

Host: server.example.com
Processing

- AS renders user consent based on type and content of the authorization data objects
- Authorization details are passed to RSs (via Access Token or Token Introspection Response)
- Parameter “resource” (draft-ietf-oauth-resource-indicators) is used by client to obtain RS-specific Access Tokens associated with the RS-specific authorization data objects only
Advantages

- Versatile and type safe
- Data structures can be optimised for resource server/API/ use case - no “one size fits all”
- Common data set elements to address common use cases
- Explicit assignment of permissions to resource servers (robust and explicit audience restriction)
- Interoperable and easy way to issue RS-specific Access Tokens and Token Introspections Responses (Data Minimisation and Disambiguation)