Pushed Request Objects

Daniel Fett, Torsten Lodderstedt, Brian Campbell

IETF-105

Pushed Request Objects

- Extension of JWT Secured Authorization Request (draft-ietf-oauth-jwsreq) (JAR) that **moves request object management to the AS**
- Specification currently being worked on in the **OpenID Foundation's FAPI WG** based on experiences gathered in OpenBanking/PSD2

https://bitbucket.org/openid/fapi/src/master/Financial_API_Pushed_Request_Object.md

Rationale

- JAR is great for ensuring integrity, authenticity, and confidentiality of authorization requests but also has some drawbacks
- `request`
  - might result in lengthy urls
- `request_uri`
  - client needs to handle inbound requests from the AS
  - client needs to store (a potentially large number of) objects & handle clean-up
  - availability and latency of client's backend influence authorization process
  - AS has to make outbound HTTP requests → problems of server-side request forgery

Pushed Request Object

- Moving the responsibility for managing request objects from client to AS
- New "request object endpoint":
  - Client calls this endpoint to deliver its request objects
  - Client is provided with a unique URI
  - Which is then used as `request_uri` parameter value

Two modes:

1. request object as JWT
2. "raw" request object in JSON format

Pushing the request object to the AS (JSON)

```
POST https://as.example.com/ros/ HTTP/1.1
Host: as.example.com
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3
Content-Type: application/json
Content-Length: 1288

{
    "response_type":"code",
    "client_id":"s6BhdRkqt3",
    "redirect_uri":"https://client.example.org/cb",
    "scope":"accounts",
    "state":"af0ifjsldkj",
    "code_challenge_method":"S256",
    "code_challenge":"5c305578f8f19b2dcdb6c3c955c0a...97e43917cd"
}
```

Pushing the request object to the AS (JWT)

```
POST https://as.example.com/ros/ HTTP/1.1
Host: as.example.com
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3
Content-Type: application/jwt
Content-Length: 1288

eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ew0KICJpc3MiOiA
(... abbreviated ...)
zCYIb_NMXvtTIVc1jpspnTSD7xMbpL-2QgwUsAlMGzw
```

Obtaining the request_uri

```
HTTP/1.1 201 Created
Date: Tue, 2 May 2017 15:22:31 GMT
Content-Type: application/json

{
     "iss": "https://as.example.com/",
     "aud": "s6BhdRkqt3",
     "request_uri": "urn:example:MTAyODAK",
     "exp": 1493738581
}
```

Sending the authorization request

```
https://server.example.com/authorize?
    request_uri=urn%3Aexample%3AMTAyODAK
```

Advantages

- No request object management at the client
- Deployments can choose from two options:
  - “Raw” JSON mode is easy to use
  - JWT mode with application level signing & encryption (e.g., for non-repudiation)
- Client authentication before the authorization process is started
  - Refuse unauthorized clients early
  - Authorization process may rely on client identity
- Solid foundation for conveying rich authorization requests
  (aka structured scopes)

Shall we bring this to the IETF?