OAuth 2.0 Pushed Authorization Requests
draft-lodderstedt-oauth-par-00

Abstract

This document defines the pushed authorization request endpoint, which allows clients to push the payload of an OAuth 2.0 authorization request to the authorization server via a direct request and provides them with a request URI that is used as reference to the data in a subsequent authorization request.

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/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 24, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
1. Introduction

In OAuth [RFC6749] authorization request parameters are typically sent as URI query parameters via redirection in the user-agent. This is simple but also yields challenges:

- There is no cryptographical integrity and authenticity protection, i.e. the request can be modified on its way through the user-agent and attackers can impersonate legitimate clients.

- There is no mechanism to ensure confidentiality of the request parameters.
Authorization request URLs can become quite large, especially in scenarios requiring fine-grained authorization data.

JWT Secured Authorization Request (JAR) [I-D.ietf-oauth-jwsreq] provides solutions for those challenges by allowing OAuth clients to wrap authorization request parameters in a signed, and optionally encrypted, JSON Web Token (JWT), the so-called "Request Object".

In order to cope with the size restrictions, JAR introduces the "request_uri" parameter that allows clients to send a reference to a request object instead of the request object itself.

This document complements JAR by providing an interoperable way to push the payload of a request object directly to the AS in exchange for a "request_uri".

It also allows for clients to push the form encoded authorization request parameters to the AS in order to exchange them for a request URI that the client can use in a subsequent authorization request.

For example, the following authorization request,

GET /authorize?response_type=code &client_id=s6BhdRkqt3&state=af0ifjs1dkj &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1
Host: as.example.com

could be pushed directly to the AS by the client as follows,

POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3
response_type=code &client_id=s6BhdRkqt3&state=af0ifjs1dkj &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb

The AS responds with a request URI,
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
}

which is used by the client in the subsequent authorization request as follows,

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

The pushed authorization request endpoint thus fosters OAuth security by providing all clients a simple means for an integrity protected authorization request, but it also allows clients requiring an even higher security level, especially cryptographically confirmed non-repudiation, to explicitly adopt JWT-based request objects.

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. Pushed Authorization Request Endpoint

The pushed authorization request endpoint shall be a RESTful API at the authorization server that accepts "x-www-form-urlencoded" POST requests.

The endpoint accepts the parameters defined in [RFC6749] for the authorization endpoint as well as all applicable extensions defined for the authorization endpoint. Some examples of such extensions include PKCE [RFC7636], Resource Indicators [I-D.ietf-oauth-resource-indicators], and OpenID Connect [OIDC].
The rules for client authentication as defined in [RFC6749] for token endpoint requests, including the applicable authentication methods, apply for the pushed authorization request endpoint as well. If applicable, the "token_endpoint_auth_method" client metadata parameter indicates the registered authentication method for the client to use when making direct requests to the authorization server, including requests to the pushed authorization endpoint.

Note that there's some potential ambiguity around the appropriate audience value to use when JWT client assertion based authentication is employed. To address that ambiguity the issuer identifier URL of the AS according to [RFC8414] SHOULD be used as the value of the audience. In order to facilitate interoperability the AS MUST accept its issuer identifier, token endpoint URL, or pushed authorization request endpoint URL as values that identify it as an intended audience.

2.1. Request

A client can send all the parameters that usually comprise an authorization request to the pushed authorization request endpoint. A basic parameter set will typically include:

- "response_type"
- "redirect_uri"
- "scope"
- "state"
- "code_challenge"
- "code_challenge_method"

Depending on client type and authentication method, the request might also include the "client_id" parameter. The "request_uri" authorization request parameter MUST NOT be provided in this case (see Section 3).

The client adds the parameters in "x-www-form-urlencoded" format with a character encoding of UTF-8 as described in Appendix B of [RFC6749] to the body of a HTTP POST request. If applicable, the client also adds client credentials to the request header or the request body using the same rules as for token endpoint requests.
This is illustrated by the following example

```
POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZ1ZkbUl3

response_type=code&
state=af0ifjsldkj&
client_id=s6BhdRkqt3&
redirect_uri=https%3A%2F%2Fclient.example.org%2Fcbb&
code_challenge=K2-ltc83acc4h0c9w6ESC_rEMTJ3bww-uCHaoeK1t8U&
scope=ais
```

The AS MUST process the request as follows:

1. The AS MUST authenticate the client in the same way as at the token endpoint.

2. The AS MUST validate the request the same way as at the authorization endpoint. For example, the authorization server checks whether the redirect URI matches one of the redirect URIs configured for the "client_id". It may also check whether the client is authorized for the "scope" for which it is requesting access. This validation allows the authorization server to refuse unauthorized or fraudulent requests early.

2.2. Successful Response

If the verification is successful, the server shall generate a request URI and return a JSON response that contains "request_uri" and "expires_in" members at the top level with "201 Created" HTTP response code.

- "request_uri" : The request URI corresponding to the authorization request posted. This URI is used as reference to the respective request data in the subsequent authorization request only. The way the authorization process obtains the authorization request data is at the discretion of the authorization server and out of scope of this specification. There is no need to make the authorization request data available to other parties via this URI.

- "expires_in" : A JSON number that represents the lifetime of the request URI in seconds. The request URI lifetime is at the discretion of the AS.
The "request_uri" value MUST be generated using a cryptographically strong pseudorandom algorithm such that it is computationally infeasible to predict or guess a valid value.

The "request URI" MUST be bound to the "client_id" of the client that posted the authorization request.

Since the request URI can be replayed, its lifetime SHOULD be short and preferably limited to one-time use.

The following is an example of such a response:

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

{
  "request_uri": "urn:example:bwc4JK-ESC0w8acc191e-Y1LTC2",
  "expires_in": 3600
}

2.3. Error Response

2.3.1. Error responses

2.3.1.1. Authentication required

If the client authentication fails, the authorization server shall return "401 Unauthorized" HTTP error response.

2.3.1.2. Authorization required

If the client is not authorized to perform the authorization request it wanted to post, the authorization server shall return "403 Forbidden" HTTP error response.

2.3.1.3. Invalid request

If the request object received is invalid, the authorization server shall return "400 Bad Request" HTTP error response.

2.3.1.4. Method not allowed

If the request did not use POST, the authorization server shall return "405 Method Not Allowed" HTTP error response.
2.3.1.5.  Request entity too large

If the request size was beyond the upper bound that the authorization
server allows, the authorization server shall return a "413 Request
Entity Too Large" HTTP error response.

2.3.1.6.  Too many requests

If the request from the client per a time period goes beyond the
number the authorization server allows, the authorization server
shall return a "429 Too Many Requests" HTTP error response.

3.  "request" Parameter

Clients MAY use the "request" parameter as defined in JAR to push a
request object to the AS. The rules for signing and encryption of
the request object as defined in JAR [I-D.ietf-oauth-jwsreq] apply.

Clients MUST NOT combine other authorization request parameters with
the "request" parameter at the pushed authorization request endpoint
other than the "client_id" parameter which may be a part of the
client authentication mechanism.

The following is an example of a request using a signed request
object. The client is authenticated using its client secret "BASIC"
authorization:

```
POST /as/par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZ1ZkbUl3

request=eyJraWQiOiJrMmJkYyIsImFsZyI6I1I1JTJMjU2In0.eyJpc3MiOiJzNkJoZ
FJrcXQzIwiYXVkIjoiaHR0cHM6Ly9zLiJzNyJoIjowM1w5Iiwicm93c25jbW91bnQI
OjA4ODAwMDUsIiwiYXNzX2NvbS5zY3Npc2l6ZS5jIiwiY2FwYWxpZ24IOTQ3NTY3Iiwia
```

The AS needs to take the following steps beyond the processing rules
defined in Section 2.1:

Lodderstedt, et al. Expires March 24, 2020
1. If applicable, the AS decrypts the request object as specified in JAR [I-D.ietf-oauth-jwsreq], section 6.1.

2. The AS validates the request object signature as specified in JAR [I-D.ietf-oauth-jwsreq], section 6.2.

3. If the client is a confidential client, the authorization server MUST check whether the authenticated "client_id" matches the "client_id" claim in the request object. If they do not match, the authorization server MUST refuse to process the request. It is at the authorization server’s discretion to require the "iss" claim to match the "client_id" as well.

3.1. Error responses for Request Object

This section gives the error responses that go beyond the basic Section 2.3.

3.1.1. Authentication Required

If the signature validation fails, the authorization server shall return "401 Unauthorized" HTTP error response. The same applies if the "client_id" or, if applicable, the "iss" claims in the request object do not match the authenticated "client_id".

4. Authorization Request

The client uses the "request_uri" value as returned by the authorization server as authorization request parameter "request_uri".

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

Clients are encouraged to use the request URI as the only parameter in order to use the integrity and authenticity provided by the pushed authorization request.

5. Authorization Server Metadata

If the authorization server has a pushed authorization request endpoint, it SHOULD include the following OAuth/OpenID Provider Metadata parameter in discovery responses:

"pushed_authorization_request_endpoint" : The URL of the pushed authorization request endpoint at which the client can exchange a request object for a request URI.
6. Security Considerations

6.1. Request URI Guessing

An attacker could attempt to guess and replay a valid request URI value and try to impersonate the respective client. The AS MUST consider the considerations given in JAR [I-D.ietf-oauth-jwsreq], section 10.2, clause d on request URI entropy.

6.2. Request Object Replay

An attacker could replay a request URI captured from a legitimate authorization request. In order to cope with such attacks, the AS SHOULD make the request URIs one-time use.

6.3. Client Policy Change

The client policy might change between the lodging of the request object and the authorization request using a particular request object. It is therefore recommended that the AS checks the request parameter against the client policy when processing the authorization request.

7. Acknowledgements

This specification is based on the work towards Pushed Request Objects [1] conducted at the Financial Grade API working group at the OpenID Foundation. We would like to thank the members of this WG for their valuable contributions.

We would like to thank ... for their valuable feedback on this draft.

8. IANA Considerations

...

9. References

9.1. Normative References

[I-D.ietf-oauth-jwsreq]
9.2. Informative References

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


9.3. URIs


Appendix A. Document History

[[ To be removed from the final specification ]]

-00

o first draft
Authors’ Addresses

Torsten Lodderstedt
yes.com

Email: torsten@lodderstedt.net

Brian Campbell
Ping Identity

Email: bcampbell@pingidentity.com

Nat Sakimura
Nomura Research Institute

Email: nat@sakimura.org

Dave Tonge
Moneyhub Financial Technology

Email: dave@tonge.org

Filip Skokan
Auth0

Email: panva.ip@gmail.com