Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: July 2, 2020

T. Lodderstedt
yes.com
B. Campbell
Ping Identity
N. Sakimura
Nomura Research Institute
D. Tonge
Moneyhub Financial Technology
F. Skokan
Auth0

December 30, 2019

# OAuth 2.0 Pushed Authorization Requests
**draft-ietf-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

## Copyright Notice

Table of Contents

## 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:

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

- o There is no mechanism to ensure confidentiality of the request parameters.

o  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=af0ifjsldkj
   &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=af0ifjsldkj
  &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 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.

As a further benefit, the pushed authorization request allows the AS to authenticate the clients before any user interaction happens, i.e., the AS may refuse unauthorized requests much earlier in the process and has much higher confidence in the client's identity in the authorization process than before.

This is directly utilized by this draft to allow confidential clients to set the redirect URI for every authorization request, which gives them more flexibility in building redirect URI.  And if the client IDs and credentials are managed by some external authority (e.g. a certification authority), explicit client registration with the particular AS could practically be skipped.

## 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 is an HTTP API at the authorization server that accepts POST requests with parameters in the HTTP request entity-body using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 as described in Appendix B of [RFC6749].

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 request 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:

o "client_id"

o "response_type"

o "redirect_uri"

o "scope"

o "state"

o  "code_challenge"

o  "code_challenge_method"

Depending on client type and authentication method, the request might also include other parameters for client authentication such as the "client_secret" parameter, the "client_assertion" parameter and the "client_assertion_type" 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 an 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 czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

  response_type=code&
  state=af0ifjsldkj&
  client_id=s6BhdRkqt3&
  redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb&
  code_challenge=K2-ltc83acc4h0c9w6ESC_rEMTJ3bww-uCHaoeK1t8U&
  code_challenge_method=S256&
  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 reject the request if the "request_uri" authorization request parameter is provided.

3.  The AS MUST validate the request in 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.  It MUST 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.

The AS MAY allow confidential clients to establish per-authorization request redirect URIs with every pushed authorization request. This is possible since, in contrast to [RFC6749], this specification gives the AS the ability to authenticate and authorize clients before the actual authorization request is performed.

This feature gives clients more flexibility in building redirect URIs and, if the client IDs and credentials are managed by some authority (CA or other type), the explicit client registration with the particular AS (manually or via dynamic client registration [RFC7591]) could practically be skipped. This makes this mechanism especially useful for clients interacting with a federation of ASs (or OpenID Connect OPs), for example in Open Banking, where the certificate is provided as part of a federated PKI.

## 2.2. Successful Response

If the verification is successful, the server MUST 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.

o "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.

o "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 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
Content-Type: application/json
Cache-Control: no-cache, no-store

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

## 2.3. Error Response

For an error the authorization server sets an appropriate HTTP status code and MAY include additional error parameters in the entity-body of the HTTP response using the format specified for the token endpoint in Section 5.2 of [RFC6749].

If the authorization server sets an error code, it SHOULD be one of the defined codes for the token endpoint in Section 5.2 or for the authorization endpoint in Sections 4.1.2.1 and 4.2.2.1 of [RFC6749], or by an OAuth extension if one is involved in the initial processing of authorization request that was pushed.  Since initial processing of the pushed authorisation request doesn't involve resource owner interaction, error codes related to user interaction, such as "consent_required" defined by [OIDC], are not returned.

In addition to the error codes above, the pushed authorization request endpoint specifies use of the following HTTP status codes:

o  405: If the request did not use POST, the authorization server responds with an HTTP 405 (Method Not Allowed) status code.

o  413: If the request size was beyond the upper bound that the authorization server allows, the authorization server responds with an HTTP 413 (Payload Too Large) status code.

o  429: If the request from the client for a time period goes beyond the number the authorization server allows, the authorization server responds with an HTTP 429 (Too Many Requests) status code.

The following is an example of an error response from the pushed authorization request endpoint:

```
     HTTP/1.1 400 Bad Request
     Content-Type: application/json
     Cache-Control: no-cache, no-store

     {
       "error": "invalid_request",
       "error_description":
         "The redirect_uri is not valid for the given client"
     }
```

## 3. "request" Parameter

Clients MAY use the "request" parameter as defined in JAR [I-D.ietf-oauth-jwsreq] to push a request object JWT to the AS. The rules for processing, signing, and encryption of the request object as defined in JAR [I-D.ietf-oauth-jwsreq] apply. When the "application/x-www-form-urlencoded" HTTP entity-body "request" parameter is used, the request object MUST contain all the authorization request parameters as claims of the JWT. Additional request parameters as required by the given client authentication method are to be included as 'application/x-www-form-urlencoded' parameters in the HTTP request entity-body (e.g. Mutual TLS client authentication [I-D.ietf-oauth-mtls] uses the "client_id" HTTP request parameter while JWT assertion based client authentication [RFC7523] uses "client_assertion" and "client_assertion_type").

The following is an example of a pushed authorization request using a signed request object. The client is authenticated by its client secret using the HTTP Basic Authentication scheme specified in Section 2.3.1 of [RFC6749]:

```
     POST /as/par HTTP/1.1
     Host: as.example.com
     Content-Type: application/x-www-form-urlencoded
     Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

     request=eyJraWQiOiJrMmJkYyIsImFsZyI6IlJTMjU2In0.eyJpc3MiOiJzNkJoZ
     FJrcXQzIiwiYXVkIjoiaHR0cHM6Ly9zZXJ2ZXIuZXhhbXBsZS5jb20iLCJyZXNwb2
     5zZV90eXBlIjoiY29kZSIsImNsaWVudF9pZCI6InM2QmhkUmtxdDMiLCJyZWRpcmV
     jdF91cmkiOiJodHRwczovL2NsaWVudC5leGFtcGxlLm9yZy9jYiIsInNjb3BlIjoi
     YWlzIiwic3RhdGUiOiJhZjBpZmpzbGRraiIsImNvZGVfY2hhbGxlbmdlIjoiSzItb
     HRjODNhY2M0aDBjOXc2RVNDX3JFTVRKM2J3dy11Q0hhb2VLMXQ4VSIsImNvZGVfY2
     hhbGxlbmdlX21ldGhvZCI6IlMyNTYifQ.O49ffUxRPdNkN3TRYDvbEYVr1CeAL64u
     W4FenV3n9WlaFIRHeFblzv-wlEtMm8-tusGxeE9z3ek6FxkhvvLEqEpjthXnyXqqy
     Jfq3k9GSf5ay74ml_0D6lHE1hy-kVWg7SgoPQ-GB1xQ9NRhF3EKS7UZIrUHbFUCF0
     MsRLbmtIvaLYbQH_Ef3UkDLOGiU7exhVFTPeyQUTM9FF-u3K-zX-FO05_brYxNGLh
     VkO1G8MjqQnn2HpAzlBd5179WTzTYhKmhTiwzH-qlBBI_9GLJmE3KOipko9TfSpa2
     6H4JOlMyfZFl0PCJwkByS0xZFJ2sTo3Gkk488RQohhgt1I0onw
```

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

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 returns a "401 Unauthorized" HTTP error response. The same applies if the "client_id" or, if applicable, the "iss" claim in the request object do not match the authenticated "client_id".

## 4. Authorization Request

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

```
  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 post an authorization request and get a request URI in exchange.

## 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. Open Redirection

An attacker could try register a redirect URI pointing to a site under his control in order to obtain authorization codes or lauch other attacks towards the user. The AS MUST only accept new redirect URIs in the PAR request from confidential clients after sucessful authentication and authorization.

### 6.3. 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.

### 6.4. 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 check the request parameter against the client policy when processing the authorization request.

## 7. Acknowledgements

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

We would like to thank Vladimir Dzhuvinov, Aaron Parecki, Joseph Heenan, and Takahiko Kawasaki for their valuable feedback on this draft.

## 8. IANA Considerations

...

## 9. References

### 9.1. Normative References

[I-D.ietf-oauth-jwsreq]
Sakimura, N. and J. Bradley, "The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)", draft-ietf-oauth-jwsreq-20 (work in progress), October 2019.

[OIDC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", Nov 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

### 9.2. Informative References

[I-D.ietf-oauth-mtls]
Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", draft-ietf-oauth-mtls-17 (work in progress), August 2019.

[I-D.ietf-oauth-resource-indicators]
Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", draft-ietf-oauth-resource-indicators-08 (work in progress), September 2019.

[RFC7523] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, <https://www.rfc-editor.org/info/rfc7523>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/info/rfc7591>.

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/info/rfc7636>.

## 9.3. URIs

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

## Appendix A. Document History

[[ To be removed from the final specification ]]

-00 (WG draft)

- o Reference RFC6749 sec 2.3.1 for client secret basic rather than RFC7617

- o further clarify that a request object JWT contains all the authorization request parameters while client authentication params, if applicable, are outside that JWT as regular form encoded params in HTTP body

-01

- o List "client_id" as one of the basic parameters

- o Explicitly forbid "request_uri" in the processing rules

- o Clarification regarding client authentication and that public clients are allowed

- Added option to let clients register per-authorization request redirect URIs

- General clean up and wording improvements

-00

- 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