Workgroup: Web Authorization Protocol
Internet-Draft: draft-ietf-oauth-par-03
Published: 31 July 2020
Intended Status: Standards Track
Expires: 1 February 2021
Authors: T. Lodderstedt B. Campbell N. Sakimura
yes.com Ping Identity NAT.Consulting
D. Tonge F. Skokan
Moneyhub Financial Technology Auth0

OAuth 2.0 Pushed Authorization Requests

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 1 February 2021.

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:

- There is no cryptographic integrity and authenticity protection. An attacker could, for example, modify the ACR value requested by the client or swap the context of a payment transaction authorization by changing scope values. Although clients should detect such changes by inspecting the token response data, preventing such modifications early in the process would be a better solution.

- There is no mechanism to ensure confidentiality of the request parameters. This obviously is an issue if personal identifiable information is sent in the authorization request, which might be the case in identity and open banking scenarios.

*Authorization request URLs can become quite large, especially in scenarios requiring fine-grained authorization data, which might cause errors in request processing.

JWT Secured Authorization Request (JAR) [I-D.ietf-oauth-jwsreq] provides solutions for the security 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 authorization server in exchange for a request_uri.

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

For example, a client typically initiates an authorization request by directing the user-agent to make an HTTP request like the following:

```
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
```

Such a request could instead be pushed directly to the authorization server 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 authorization server 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
}
```

The client uses the request URI value to create the subsequent authorization request and directing the user-agent to make an HTTP request like the following:

```
GET /authorize?client_id=s6BhdRkqt3&
request_uri=urn%3Aexample%3Abwc4JK-ESC0w8acc191e-Y1LTC2 HTTP/1.1
Host: as.example.com
```

The pushed authorization request endpoint fosters OAuth security by providing all clients a simple means for a confidential and 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 authorization server to authenticate the clients before any user interaction happens, i.e., the authorization server 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 generally improves security since it prevents attempts to spoof confidential clients early in the process.

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 authorization server could practically be skipped.

Note: HTTP POST requests to the authorization endpoint as described in Section 3.1 of [RFC6749] and Section 3.1.2.1 of [OIDC] could also be used to cope with the request size limitations described above. Although this is a viable option for traditional web applications, it's difficult to use with mobile apps. Those apps typically invoke a custom tab with an URL that is translated into a GET request. Using POST would require the app to first open a web page under its control in the custom tab that in turn would initiate the form POST

towards the authorization server. PAR is simpler to use and has additional security benefits as described above.

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 pushed authorization request endpoint URL MUST use the "https" scheme.

Authorization servers supporting pushed authorization requests SHOULD include the URL of their pushed authorization request endpoint in their authorization server metadata document [RFC8414] using the pushed_authorization_request_endpoint parameter as defined in Section 5.

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 [RFC8707], and OpenID Connect [OIDC]. The endpoint also supports sending all authorization request parameters as request object according to [I-D.ietf-oauth-jwsreq].

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 authorization server according to [RFC8414] SHOULD be used as the value of the audience. In order to facilitate interoperability the authorization server 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:

* client_id
* response_type
* redirect_uri
* scope
* state
* code_challenge
* 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 authorization server MUST process the request as follows:

1. Authenticate the client in the same way as at the token endpoint.

2. Reject the request if the request_uri authorization request parameter is provided.

3. Validate the pushed request as it would an authorization request sent to the authorization endpoint. For example, the authorization server checks whether the redirect URI matches one of the redirect URIs configured for the client and also checks 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 authorization server MAY omit validation steps that it is unable to perform when processing the pushed request, however such checks MUST then be performed at the authorization endpoint.

The authorization server 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 authorization server 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 authorization server (manually or via dynamic client registration [RFC7591]) could practically be skipped. This makes this mechanism especially useful for clients interacting with a federation of authorization servers (or OpenID Connect Providers), 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 with the following 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 authorization server and will typically be relatively short.

The format of the request_uri value is at the discretion of the authorization server but it MUST contain some part generated using a cryptographically strong pseudorandom algorithm such that it is computationally infeasible to predict or guess a valid value. The authorization server MAY construct the request_uri value using the form urn:ietf:params:oauth:request_uri:<reference-value> with <reference-value> as the random part of the URI that references the respective authorization request data. The string representation of a UUID as a URN per [RFC4122] is also an option for authorization servers to construct request_uri values.

The request_uri MUST be bound to the client that posted the authorization request.

Since parts of the request content, e.g. the code_challenge parameter value, is unique to a certain authorization request, a request_uri SHOULD be 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:ietf:params:oauth:request_uri:bwc4JK-ESC0w8acc191e-Y1LTC2",
  "expires_in": 60
}
```

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 authorization request doesn't involve resource owner interaction, error codes related to user interaction, such as consent_required defined by [OIDC], are not returned.

If the client is required to use signed request objects, either by authorization server or client policy (see [I-D.ietf-oauth-jwsreq], section 10.5), the authorization server MUST only accept requests complying with the definition given in Section 3 and MUST refuse any other request with HTTP status code 400 and error code invalid_request.

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

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

- 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.

- 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 authorization server. 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 authorization server needs to take the following steps beyond the processing rules defined in Section 2.1:

1. If applicable, decrypt the request object as specified in JAR [I-D.ietf-oauth-jwsreq], section 6.1.

2. 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.

The following RSA key pair, represented in JWK [RFC7517] format, can be used to validate or recreate the request object signature in the above example (line wraps within values for display purposes only):

```
{
  "kty": "RSA",
  "kid":"k2bdc",
  "n": "y9Lqv4fCp6Ei-u2-ZCKq83YvbFEk6JMs_pSj76eMkddWRuWX2aBKGHAtKlE
        5P7_vn__PCKZWePt3vGkB6ePgzAFu08NmKemwE5bQI0e6kIChtt_6KzT5Oa
        aXDFI6qCLJmk51Cc4VYFaxgqevMncYrzaW_50mZ1yGSFIQzLYP8bijAHGVj
        dEFgZaZEN9lsn_GdWLaJpHrB3ROlS50E45wxrlg9xMncVb8qDPuXZarvghL
        L0HzOuYRadBJVoWZowDNTpKpk2RklZ7QaBO7XDv3uR7s_sf2g-bAjSYxYUG
        sqkNA9b3xVW53am_UZZ3tZbFTIh557JICWKHlWj5uzeJXaw",
  "e": "AQAB",
  "d": "LNwG_pCKrwowALpCpRdcOKlSVqylSurZhE6CpkRiE9cpDgGKIkO9CxPlXOL
        zjqxXuQc8MdMqRQZTnAwgd7HH0B6gncrruV3NewI-XQV0ckldTjqNfOTz1V
        Rs-jE-57KAXI3YBIhu-_0YpIDzdk_wBuAk661Svn0GsPQe7m9DoxdzenQu9
        O_soewUhlPzRrTH0EeIqYI715rwI3TYaSzoWBmEPD2fICyj18FF0MPy_SQz
        k3noVUUIzfzLnnJiWy_p63QBCMqjRoSHHdMnI4z9iVpIwJWQ3jO5n_2lC2-
        cSgwjmKsFzDBbQNJc7qMG1N6EssJUwgGJxz1eAUFf0w4YAQ",
  "qi": "J-mG0swR4FTy3atrcQ7dd0hhYn1E9QndN-
        -sDG4EQO0RnFj6wIefCvwIc4
        7hCtVeFnCTPYJNc_JyV-mU-9vlzS5GSNuyR5qdpsMZXUMpEvQcwKt23ffPZ
        YGaqfKyEesmf_Wi8fFcE68H9REQjnniKrXm7w2-IuG_IrVJA9Ox-uU",
  "q": "4hlMYAGa0dvogdK1jnxQ7J_Lqpqi99e-AeoFvoYpMPhthChTzwFZO9lQmUo
        BpMqVQTws_s7vWGmt7ZAB3ywkurf0pV7BD0fweJiUzrWk4KJjxtmP_auuxr
        jvm3s2FUGn6f0wRY9Z8Hj9A7C72DnYCjuZiJQMYCWDsZ8-d-L1a-s",
  "p": "5sd9Er3I2FFT9R-gy84_oakEyCmgw036B_nfYEEOCwpSvi2z7UcIVK3bSEL
        5WCW6BNgB3HDWhq8aYPirwQnqm0K9mX1E-4xM10WWZ-rP3XjYpQeS0Snru5
        LFVWsAzi-FX7BOqBibSAXLdEGXcXa44l08iec_bPD3xduq5V_1YoE",
  "dq": "Nz2PF3XM6bEc4XsluKZO70ErdYdKgdtIJReUR7Rno_tOZpejwlPGBYVW19
        zpAeYtCT82jxroB2XqhLxGeMxEPQpsz2qTKLSe4BgHY2ml2uxSDGdjcsrbb
        NoKUKaN1CuyZszhWl1n0AT_bENl4bJgQj_Fh0UEsQj5YBBUJt5gr_k",
  "dp": "Zc877jirkkLOtyTs2vxyNe9KnMNAmOidlUc2tE_-0gAL4Lpo1hSwKCtKwe
        ZJ-gkqt1hT-dwNx_0Xtg_-NXsadMRMwJnzBMYwYAfjApUkfqABc0yUCJJl3
        KozRCugf1WXkU9GZAH2_x8PUopdNUEa70ISowPRh04HANKX4fkjWAE"
 }
```

4. Authorization Request

The client uses the request_uri value returned by the authorization server to build an authorization request as defined in [I-D.ietf-oauth-jwsreq]. This is shown in the following example where the client directs the user-agent to make the following HTTP request:

```
GET /authorize?client_id=s6BhdRkqt3&request_uri=urn%3Aietf%3Aparams
%3Aoauth%3Arequest_uri%3Abwc4JK-ESC0w8acc191e-Y1LTC2 HTTP/1.1
Host: as.example.com
```

The authorization server MUST validate authorization requests arising from a pushed request as it would any other authorization request. The authorization server MAY omit validation steps that it performed when the request was pushed, provided that it can validate that the request was a pushed request, and that the request or the authorization server's policy has not been modified in a way that would affect the outcome of the omitted steps.

Authorization server policy MAY dictate, either globally or on a per-client basis, that pushed authorization requests are the only means for a client to pass authorization request data. In this case, the authorization server will refuse, using the invalid_request error code, to process any request to the authorization endpoint that does not have a request_uri parameter with a value obtained from the pushed authorization request endpoint.

Note: authorization server and clients MAY use metadata as defined in Section 5 and Section 6 to signal the desired behavior.

5. Authorization Server Metadata

The following authorization server metadata [RFC8414] parameters are introduced to signal the server's capability and policy with respect to pushed authorization requests.

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.

require_pushed_authorization_requests Boolean parameter indicating whether the authorization server accepts authorization request data only via the pushed authorization request method. If omitted, the default value is false.

6. Client Metadata

The Dynamic Client Registration Protocol [RFC7591] defines an API for dynamically registering OAuth 2.0 client metadata with authorization servers. The metadata defined by [RFC7591], and registered extensions to it, also imply a general data model for clients that is useful for authorization server implementations even when the Dynamic Client Registration Protocol isn't in play. Such implementations will typically have some sort of user interface available for managing client configuration. The following client metadata parameter is introduced by this document to indicate

whether pushed authorization requests are reqired for the given client.

require_pushed_authorization_requests Boolean parameter indicating whether the only means of initiating an authorization request the client is allowed to use is a pushed authorization request.

7. Security Considerations

7.1. Request URI Guessing

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

7.2. 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 authorization server MUST only accept new redirect URIs in the PAR request from confidential clients after successful authentication and authorization.

7.3. Request Object Replay

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

7.4. 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 authorization server check the request parameter against the client policy when processing the authorization request.

8. Acknowledgements

This specification is based on the work towards Pushed Request Object 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, Justin Richer, Sascha Preibisch, Daniel Fett, Michael B. Jones, Annabelle Backman, Joseph Heenan, Sean Glencross, Maggie Hung, Neil Madden, and Takahiko Kawasaki for their valuable feedback on this draft.

9. IANA Considerations

9.1. OAuth Authorization Server Metadata

This specification requests registration of the following values in the IANA "OAuth Authorization Server Metadata" registry of [IANA.OAuth.Parameters] established by [RFC8414].

Metadata Name: pushed_authorization_request_endpoint
Metadata Description: URL of the authorization server's pushed authorization request endpoint
Change Controller: IESG
Specification Document(s): Section 5 of [[this document]]
Metadata Name: require_pushed_authorization_requests
Metadata Description: Indicates whether the authorization server accepts authorization request only via the pushed authorization request method.
Change Controller: IESG
Specification Document(s): Section 5 of [[this document]]

9.2. OAuth Dynamic Client Registration Metadata

This specification requests registration of the following value in the IANA "OAuth Dynamic Client Registration Metadata" registry of [IANA.OAuth.Parameters] established by [RFC7591].

Metadata Name: require_pushed_authorization_requests
Metadata Description: Indicates whether the client is required to use the pushed authorization request method to initiate authorization requests.
Change Controller: IESG
Specification Document(s): Section 6 of [[this document]]

9.3. OAuth URI Registration

This specification requests registration of the following value in the "OAuth URI" registry of [IANA.OAuth.Parameters] established by [RFC6755].

URN: urn:ietf:params:oauth:request_uri:
Common Name: A URN Sub-Namespace for OAuth Request URIs.
Change Controller: IESG
Specification Document(s): Section 2.2 of [[this document]]

10. Normative References

[I-D.ietf-oauth-jwsreq] Sakimura, N. and J. Bradley, "The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)", Work in Progress, Internet-Draft, draft-

ietf-oauth-jwsreq-26, 27 July 2020, <https://tools.ietf.org/html/draft-ietf-oauth-jwsreq-26>.

[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>.

[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>.

[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>.

[OIDC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[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>.

11. Informative References

[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>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[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", Work in Progress, Internet-Draft, draft-ietf-oauth-mtls-17, 23 August 2019, <https://tools.ietf.org/html/draft-ietf-oauth-mtls-17>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[IANA.OAuth.Parameters]
IANA, "OAuth Parameters", , <http://www.iana.org/assignments/oauth-parameters>.

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, DOI 10.17487/RFC6755, October 2012, <https://www.rfc-editor.org/info/rfc6755>.

[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>.

[RFC8707] Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/info/rfc8707>.

[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>.

Appendix A. Document History

[[To be removed from the final specification]]

-03

* Editorial updates

* Mention that https is required for the PAR endpoint

* Add some discussion of browser form posting an authz request vs. the benefits of PAR for any application

* Added text about motivations behind PAR - integrity, confidentiality and early client auth

* Better explain one-time use recommendation of the request_uri

* Drop the section on special error responses for request objects

* Clarify authorization request examples to say that the client directs the user-agent to make the HTTP GET request (vs. making the request itself)

-02

* Update Resource Indicators reference to the somewhat recently published RFC 8707

* Added metadata in support of pushed authorization requests only feature

* Update to comply with draft-ietf-oauth-jwsreq-21, which requires client_id in the authorization request in addition to the request_uri

* Clarified timing of request validation

* Add some guidance/options on the request URI structure

* Add the key used in the request object example so that a reader could validate or recreate the request object signature

* Update to draft-ietf-oauth-jwsreq-25 and added note regarding require_signed_request_object

-01

* Use the newish RFC v3 XML and HTML format
* Added IANA registration request for pushed_authorization_request_endpoint
* Changed abbrev to "OAuth PAR"

-00 (WG draft)

* Reference RFC6749 sec 2.3.1 for client secret basic rather than RFC7617
* 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

* List client_id as one of the basic parameters
* Explicitly forbid request_uri in the processing rules
* 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
NAT.Consulting

Email: nat@sakimura.org

Dave Tonge
Moneyhub Financial Technology

Email: dave@tonge.org

Filip Skokan
Auth0

Email: panva.ip@gmail.com