WG Working Group
Internet-Draft
Intended status: Standards Track
Expires: 24 October 2025

J. Richer
Bespoke Engineering
A. Parecki
Okta

22 April 2025

OAuth Pushed Client Registration
draft-richer-oauth-pushed-client-registration-00

Abstract

This specification provides a way for an ephemeral or transactional OAuth 2.0 client to signal to the AS that the client does not need or expect to have an identified state outside the existence of any issued access and refresh tokens. The specification also enables a client to push its registration information to the AS during a pushed authorization request.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://example.com/LATEST. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-richer-oauth-pushed-client-registration/.

Discussion of this document takes place on the WG Working Group mailing list (mailto:WG@example.com), which is archived at https://example.com/WG.

Source for this draft and an issue tracker can be found at https://github.com/USER/REPO.

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/.

Copyright Notice

Table of Contents

1. Introduction

The OAuth protocol family was originally designed around a model of two relatively stable web sites connecting to each other's APIs on behalf of a user. This design, coupled with the need to begin transactions through an in-browser redirect in the authorization code and implicit grant types, leads naturally to an identifier for the client that can be presented in the front channel and referenced in the back channel along with client authentication. The client identifier could uniquely identify the client at the AS and be used

to set policy and track its requests over time.

Not all clients are sufficiently stable or monolithic to fit this model well. To compensate, OAuth 2.0 introduced the concept of "public clients", whereby many instances of a client (such as a native application or single-page browser application) would share a single identifier. OAuth Dynamic Client Registration [RFC7591] enabled client software that did not get a client identifier at configuration time to obtain an identifier at runtime through a separate request to the AS. This enables dynamic configuration, but comes at a cost of needing to securely manage issuing and tracking these additional identifiers.

Some forms of client software in open ecosystem patterns, such as the Model Context Protocol or many legacy software authentication systems like IMAP, are built around a different model whereby any compliant piece of client software should be able to connect to a compliant server, without a managed preregistration. For these clients, this specification defines a mechanism to indicate to the AS that a client does not need a longstanding identifier but instead desires authorization within a single delegation transaction.

By using OAuth Pushed Authorization Requests [RFC9126], a Pushed Client Registration can enable open world protocols to use the OAuth family of protocols as their security layer.

2. Conventions and Definitions

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 document contains non-normative examples of partial and complete HTTP messages. Some examples use a single trailing backslash () to indicate line wrapping for long values, as per [RFC8792]. The \ character and leading spaces on wrapped lines are not part of the value.

3. Protocol Flow

Use of this specification consists of the following steps:

* The client makes a pushed authorization request using a defined static client_id value

* The AS returns a request_uri which the AS can associate to the specific request

* The client makes an authorization request using the request_uri

* The AS processes the authorization request based on the parameters pushed during the pushed authorization request

* The AS creates and returns an authorization code to the client

* The client submits the authorization code to the AS, again using the static client_id value

3.1. Pushed Authorization Request and Response

The client MUST start the transaction using a pushed authorization request per [RFC9126]. The request MUST include a client_id value of urn:ietf:oauth:parameters:dynamic. The request MUST include a state value and consist of a random number. The request MUST include a redirect_urivalue. The request MUST include an OAuth PKCE request per [RFC7636].

The request MAY include a client_registration field, which consists of a form-encoded serialization of the JSON client metadata document defined in [RFC7591].

NOTE: '\' line wrapping per RFC 8792

```
POST /par HTTP/1.1
Host: as.example.com
DPoP: eyj…
DPoP-Key: ejy…

client_id=urn:ietf:oauth:parameters:dynamic\
  &response_type=code\
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb\
  &state=superrandom4789\
  &code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM\
  &code_challenge_method=S256\
  &client_registration=%7B%22client_name%22%3A%22ClaudeDesktop%22%2C%0A\
      %20%22logo_uri%22%3D%22https%3A%2F%2Fanthropic.com%2Flogo.png%22%7D
```

Where the value of 'client_registration_ is the form-encoded JSON value:

```
{
  "client_name":"ClaudeDesktop",
  "logo_uri"="https://anthropic.com/logo.png"
}
```

If the client intends to use a sender-contstrained OAuth access token, such as via DPoP [RFC9449] or mTLS [RFC8705], the client MUST send the bound authentication parameters appropriate to the method to the PAR endpoint, as in the above example.

The AS creates a pending authorization request and associates it with an internal identifier, which it returns as a URN to the client:

```
HTTP 200 OK

{
   "request_uri": "urn:example:g012h340-4j138fha",
   "expires_in": 90
 }
```

3.2. Authorization Request and Response

The client takes the result of the pushed authorization request and uses it to create an authorization request to the AS. The client MUST NOT include any parameters other than the syntactically required client_id, which is set to the static value in Section 4.

NOTE: '\' line wrapping per RFC 8792

```
GET /authorize?client_id=urn:ietf:oauth:parameters:dynamic\
  &request_uri=urn%3Aexample%3Ag012h340-4j138fha HTTP/1.1
Host: as.example.com
```

The AS parses the value of the request_uri parameter and associates the incoming request with the values saved during the call to the PAR endpoint.

When the AS has received appropriate authorization from the RO, the details of which are out of scope of this specification, the AS returns an authorization code to the client, along with its initial state value.

NOTE: '\' line wrapping per RFC 8792

```
HTTP 301 Found
Location: https://client.example.org/cb?\
  code=98765cd-oe\
  &state=superrandom4789
```

When the client receives this request, the client compares the value of state to the stored state value to ensure they are equal to the one the client is expecting.

3.3. Toiken Request and Response

The client takes the authorization code and sends it to the AS in exchange for a token. In this example, the client is requesting a DPoP bound access token.

NOTE: '\' line wrapping per RFC 8792

```
POST /token HTTP/1.1
Host: as.example.com
Content-type: application/www-form-urlencoded
DPoP: eyj…
DPoP-Key: ejy…

client_id=urn:ietf:oauth:parameters:dynamic\
  &code=98765cd-oe\
  &grant_type=authorization_code\
  &code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

The AS processes the authorization code and issues an access token to the client.

NOTE: '\' line wrapping per RFC 8792

```
HTTP 200 OK
Content-Type: application/json

{
  "access_token": "654edfgh-jkoiu789.0plkjhg34k",
  "token_type": "DPoP"
}
```

The client can then use this token to call the RS.

3.4. Token Expiration and Refresh

The AS MAY issue a refresh token to the client, which can be used to get a new access token with the exact access rights of the original request.

When the access token expires and the client has no valid refresh token, the client MUST create a new pushed authorization request to get a new token.

4. Client Identifier

The client MUST use a client_id value of urn:ietf:oauth:parameters:dynamic in all requests.

When receiving a client_id value of urn:ietf:oauth:parameters:dynamic, the AS MUST process it according to this specification. An AS MUST NOT assign the value urn:ietf:oauth:parameters:dynamic to any single client registration.

5. Discoverable Support

An AS supporting this extension that publishes its metadata using [RFC8414] MUST indicate its support by including the key:

"pushed_client_registration_supported": A boolean value indicating that this specification is supported and accepted by the AS.

6. Security Considerations

Support for this specification does not imply that it is supported for all possible access requests. In fact, an AS would be expected to determine through policy and configuration whether the request being made is suitable for a pushed client.

7. IANA Considerations

* Register the "dynamic" value in the IETF OAuth URN namespace

* register the "client_registration" authorization parameter

* extend the parameters table to indicate that a parameter can only be used inside a PAR request?

* register the AS metadata key

8. Normative References

[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/rfc/rfc2119>.

[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/rfc/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/rfc/rfc7636>.

[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/rfc/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/rfc/rfc8414>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/rfc/rfc8705>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/rfc/rfc8792>.

[RFC9126] Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, <https://www.rfc-editor.org/rfc/rfc9126>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/rfc/rfc9449>.

Acknowledgments

The authors would like to thank Darin McAdams, Den Delimarsky, David Soria Perra, Adrian Frei, Pieter Kasselman, Jerome Swannack, Basil Hosmer, and Fei Yuan for their discussion and input to the concepts behind this draft.

Authors' Addresses

Justin Richer
Bespoke Engineering
Email: ietf@justin.richer.org
URI: https://bspk.io/

Aaron Parecki
Okta
Email: aaron@parecki.com
URI: https://aaronparecki.com