Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 10, 2013

J. Richer, Ed.
The MITRE Corporation
J. Bradley
Ping Identity
M. Jones
Microsoft
M. Machulak
Newcastle University

February 6, 2013

OAuth Dynamic Client Registration Protocol
draft-ietf-oauth-dyn-reg-05

Abstract

This specification defines an endpoint and protocol for dynamic registration of OAuth Clients at an Authorization Server.

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 http://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 August 10, 2013.

Table of Contents

1. Introduction

In some use-case scenarios, it is desirable or necessary to allow OAuth clients to obtain authorization from an OAuth authorization server without requiring the two parties to interact before hand. Nevertheless, in order for the authorization server to accurately and securely represent to end-users which client is seeking authorization to access the end-user's resources, a method for automatic and unique registration of clients is needed. The OAuth2 authorization framework does not define how the relationship between the Client and the Authorization Server is initialized, or how a given client is assigned a unique Client Identifier. Historically, this has happened out-of-band from the OAuth protocol. This draft provides a mechanism for a client to register itself with the Authorization Server, which can be used to dynamically provision a Client Identifier, and optionally a Client Secret.

As part of the registration process, this specification also defines a mechanism for the client to present the Authorization Server with a set of metadata, such as a display name and icon to be presented to the user during the authorization step. This draft provides a method for the client to register and update this information over time.

1.1. Notational Conventions

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in [RFC2119].

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

1.2. Terminology

This specification uses the terms "Access Token", "Refresh Token", "Authorization Code", "Authorization Grant", "Authorization Server", "Authorization Endpoint", "Client", "Client Identifier", "Client Secret", "Protected Resource", "Resource Owner", "Resource Server", and "Token Endpoint" defined by OAuth 2.0 [RFC6749].

This specification defines the following additional terms:

- Client Registration Endpoint: The OAuth 2.0 Endpoint through which a Client can request new registration.

- Client Update Endpoint: The OAuth 2.0 Endpoint through which a specific Client can manage its registration information, provided by the Authorization Server to the Client.

- Client Secret Rotation Endpoint: The OAuth 2.0 Endpoint through which a specific Client can request refreshes of its Client Secret and Registration Access Token.

- Registration Access Token: An OAuth 2.0 Bearer Token issued by the Authorization Server through the Client Registration Endpoint which is used by the Client to authenticate itself during update and secret rotation operations. This token is associated with a particular Client.

2. Client Metadata

Clients generally have an array of metadata associated with their unique Client Identifier at the Authorization Server. These can range from human-facing display strings, such as a client name, to items that impact the security of the protocol, such as the list of valid redirect URIs.

Extensions and profiles of this specification MAY expand this list, but MUST at least accept all parameters on this list. The Authorization Server MUST ignore any additional parameters sent by the Client that it does not understand.

[[Editor's note: normative language in the table below is meant to apply to the *client* when sending the request. The paragraph above is meant to say that the server must at least accept all parameters and not fail with an error at an unknown parameter, especially if it's in the list below. Also, extensions need to explicitly call out if they're not going to do something with one of these basic parameters instead of just ignoring their existence. This is meant to be the *minimum set* of parameters for interoperability.]]

redirect_uris
: RECOMMENDED. A list of redirect URIs for use in the Authorization Code and Implicit grant types. An Authorization Server SHOULD require registration of valid redirect URIs for all clients that use these grant types in order to protect against token and credential theft attacks.

client_name
: RECOMMENDED. Human-readable name of the Client to be presented to the user. If omitted, the Authorization Server MAY display to the user the raw "client_id" value instead.

client_url
: RECOMMENDED. URL of the homepage of the Client. If present, the server SHOULD display this URL to the end user in a clickable fashion.

logo_url
: OPTIONAL. URL that references a logo for the Client. If present, the server SHOULD display this image to the end user during approval.

contacts
: OPTIONAL. List of email addresses for people responsible for this Client. The Authorization Server MAY make these addresses available to end users for support requests for the Client. An Authorization Server MAY use these email addresses as identifiers for an administrative page for this client.

tos_url
: OPTIONAL. URL that points to a human-readable Terms of Service for the Client. The Authorization Server SHOULD display this URL to the End-User if it is given.

token_endpoint_auth_method
: OPTIONAL. The requested authentication type for the Token Endpoint. Valid values are:

  * "none": this is a public client as defined in OAuth 2.0 and does not have a client secret

  * "client_secret_post": the client uses the HTTP POST parameters defined in OAuth2.0 section 2.3.1

  * "client_secret_basic": the client uses HTTP Basic defined in OAuth 2.0 section 2.3.1

  * "client_secret_jwt": the client uses the JWT Assertion profile with a symmetric secret issued by the server

  * "private_key_jwt": the client uses the JWT Assertion profile with its own private key

  Other authentication methods may be defined by extension. If unspecified or omitted, the default is "client_secret_basic", denoting HTTP Basic Authentication Scheme as specified in Section 2.3.1 of OAuth 2.0.

scope
: OPTIONAL. Space separated list of scopes (as described in OAuth 2.0 Section 3.3 [RFC6749]) that the client will be allowed to request tokens for. If omitted, an Authorization Server MAY register a Client with a default set of allowed scopes.

grant_type
: OPTIONAL. List of grant types that a client may use. These grant types are defined as follows:

  * "authorization_code": The Authorization Code Grant described in OAuth2 Section 4.1.

  * "implicit": The Implicit Grant described in OAuth2 Section 4.2.

  * "password": The Resource Owner Password Credentials Grant described in OAuth2 Section 4.3

  * "client_credentials": The Client Credentials Grant described in OAuth2 Section 4.4

  * "refresh_token": The Refresh Token Grant described in OAuth2 Section 6.

  Authorization Servers MAY allow for other values as defined in grant type extensions to OAuth2. The extension process is described in OAuth2 Section 2.5, and the value of this parameter MUST be the same as the value of the "grant_type" parameter defined in the extension.

policy_url
: OPTIONAL. A URL location that the Client provides to the End-User to read about the how the profile data will be used. The Authorization Server SHOULD display this URL to the End-User if it is given.

jwk_url
: OPTIONAL. URL for the Client's JSON Web Key [JWK] document that is used for signing requests, such as requests to the Token Endpoint using the "private_key_jwt" assertion client credential. If the Client registers both "x509_url" and "jwk_url", the keys contained in both formats MUST be the same.

jwk_encryption_url
: OPTIONAL. URL for the Client's JSON Web Key [JWK] that the server can use to encrypt responses to the Client. If the Client registers both "jwk_encryption_url" and "x509_encryption_url", the keys contained in both formats MUST be the same.

x509_url
: OPTIONAL. URL for the Client's PEM encoded X.509 Certificate or Certificate chain that is used for signing requests, such as requests to the Token Endpoint using the "private_key_jwt" assertion client credential. If the Client registers both "x509_url" and "jwk_url", the keys contained in both formats MUST be the same.

x509_encryption_url
: OPTIONAL. URL for the Client's PEM encoded X.509 Certificate or Certificate chain that the server can use to encrypt responses to the Client. If the Client registers both "jwk_encryption_url" and "x509_encryption_url", the keys contained in both formats MUST be the same.

3. Client Registration Endpoint

The Client Registration Endpoint is an OAuth 2.0 Endpoint defined in this document that is designed to allow a Client to register itself with the Authorization Server. The Client Registration Endpoint MUST accept HTTP POST messages with request parameters encoded in the entity body using the "application/json" format. The Client Registration Endpoint MUST be protected by a transport-layer security mechanism, and the server MUST support TLS 1.2 RFC 5246 [RFC5246] and/or TLS 1.0 [RFC2246] and MAY support additional transport-layer mechanisms meeting its security requirements. When using TLS, the Client MUST perform a TLS/SSL server certificate check, per RFC 6125 [RFC6125].

The Client Registration Endpoint MAY accept an initial authorization credential in the form of an OAuth 2.0 [RFC6749] access token in order to limit registration to only previously authorized parties. The method by which this access token is obtained by the registrant is generally out-of-band and is out of scope of this specification.

In order to support open registration and facilitate wider interoperability, the Client Registration Endpoint SHOULD allow initial registration requests with no authentication. These requests MAY be rate-limited or otherwise limited to prevent a denial-of-service attack on the Client Registration Endpoint.

In order to facilitate registered clients updating their information, the Client Registration Endpoint issues a Request Access Token for clients to securely identify themselves in future connections to the Client Update Endpoint. As such, the Client Update Endpoint MUST accept requests with OAuth 2.0 Bearer Tokens [RFC6750] for these operations, whether or not the initial registration call requires

authentication of some form.

The Client Registration Endpoint MUST ignore all parameters it does not understand.

3.1. Client Registration Request

This operation registers a new Client to the Authorization Server. The Authorization Server assigns this client a unique Client Identifier, optionally assigns a Client Secret, and associates the metadata given in the request with the issued Client Identifier. The request includes any parameters described in Client Metadata (Section 2) that the client wishes to specify for itself during the registration. The Authorization Server MAY provision default values for any items omitted in the Client Metadata.

The Client sends an HTTP POST to the Client Registration Endpoint with a content type of "application/json" and all parameters as top-level members of a JSON object.

For example, a client could send the following registration request to the Client Registration Endpoint:

Following is a non-normative example request (with line wraps for display purposes only):

```
POST /register HTTP/1.1
Accept: application/json
Host: server.example.com

{
 "redirect_uris":["https://client.example.org/callback",
    "https://client.example.org/callback2"]
 "client_name":"My Example Client",
 "token_endpoint_auth_method":"client_secret_basic",
 "scope":"read write dolphin",
 "logo_url":"https://client.example.org/logo.png",
 "jwk_url":"https://client.example.org/my_rsa_public_key.jwk"
}
```

3.2. Client Registration Response

Upon successful registration, the Client Registration Endpoint returns the newly-created Client Identifier and, if applicable, a Client Secret.

Additionally, the Authorization Server SHOULD return all registered metadata (Section 2) about this client, including any fields provisioned by the Authorization Server itself. The Authorization

Server MAY reject or replace any of the client's requested metadata values submitted during the registration request and substitute them with suitable values. If the Authorization Server performs any such substitutions to the requested values, it MUST return these values in the response.

The response contains a "_links" structure which contains fully qualified URLs to the Client Update Endpoint and the Client Secret Rotation Endpoint for this specific client. The response also contains a Registration Access Token that is to be used by the client to perform subsequent operations at the Client Update Endpoint and the Client Secret Rotation Endpoint.

The response is an "application/json" document with the following parameters in addition to any applicable client metadata fields as top-level members of a JSON object [RFC4627] .

client_id
: REQUIRED. The unique Client identifier, MUST NOT be currently valid for any other registered Client.

client_secret
: OPTIONAL. The Client secret. If issued, this MUST be unique for each "client_id". This value is used by confidential clients to authenticate to the Token Endpoint as described in OAuth 2.0 Section 2.3.1.

registration_access_token
: REQUIRED. The Access token to be used by the client to perform actions on the Client Update Endpoint and the Client Secret Rotation Endpoint.

issued_at
: OPTIONAL. Specifies the timestamp when the Client Identifier was issued. The timestamp value MUST be a positive integer. The value is expressed in the number of seconds since January 1, 1970 00:00:00 GMT.

expires_at
: REQUIRED if "client_secret" is issued. The number of seconds from 1970-01-01T0:0:0Z as measured in UTC that the "client_secret" will expire or "0" if it does not expire. See RFC 3339 [RFC3339] for details regarding date/times in general and UTC in particular.

_links
: REQUIRED. A JSON object that contains references to the Client Update Endpoint and Client Secret Rotation Endpoint, via the following members:

self REQUIRED. A JSON object that contains the member href which contains the fully qualified URL of the Client Update Endpoint for this client. This MAY be constructed using a URL Template of the Client Registration Endpoint with the issued client_id.

rotate_secret REQUIRED. A JSON object that contains the member href which contains the fully qualified URL of the Client Secret Rotation Endpoint for this client. This MAY be constructed using a URL Template of the Client Registration Endpoint with the issued client_id.

Following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
 _links: {
   "self": {
     "href":
       "https://server.example.com/register/s6BhdRkqt3"
   },
   "rotate_secret": {
     "href":
       "https://server.example.com/register/rotate_secret/s6BhdRkqt3"
   }
 "redirect_uris":["https://client.example.org/callback",
    "https://client.example.org/callback2"]
 "client_id":"s6BhdRkqt3",
 "client_secret": "cf136dc3c1fc93f31185e5885805d",
 "scope": "read write dolphin",
 "grant_type": ["authorization_code", "refresh_token"]
 "token_endpoint_auth_method": "client_secret_basic",
 "logo_url": "https://client.example.org/logo.png",
 "jwk_url": "https://client.example.org/my_rsa_public_key.jwk",
 "registration_access_token": "reg-23410913-abewfq.123483",
 "expires_at":2893276800
}
```

3.3. Client Registration Error Response

When an OAuth error condition occurs, the Client Registration Endpoint returns an Error Response as defined in Section 5.2 of the OAuth 2.0 specification.

When a registration error condition occurs, the Client Registration Endpoint returns a HTTP 400 status code including a JSON object [RFC4627] describing the error in the response body.

The JSON object contains two members:

error
: The error code, a single ASCII string.

error_description
: The additional text description of the error for debugging.

This specification defines the following error codes:

invalid_redirect_uri
: The value of one or more "redirect_uris" is invalid.

invalid_client_metadata
: The value of one of the client metadata (Section 2) fields is invalid and the server has rejected this request. Note that an Authorization server MAY choose to substitute a valid value for any requested parameter of a client's metadata.

Following is a non-normative example of an error response (with line wraps for display purposes only):

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

{
 "error":"invalid_redirect_uri",
 "error_description":"The redirect URI of http://sketchy.example.com
   is not allowed for this server."
}
```

4. Client Update Endpoint

The Client Update Endpoint is an OAuth 2.0 protected endpoint that is provisioned by the server for a specific client to be able to view and update its registered information. It is RECOMMENDED that this endpoint URL be formed through the use of a URL template which combines the Client Registration Endpoint and the issued client_id for this client, either as a path parameter (https://server.example.com/register/client_id) or as a query parameter (https://server.example.com/register/?update=client_id). The Authorization Server MUST provide the client with the fully qualified URL in the _links structure described in section 3 and MUST NOT require the client to construct this URL on its own.

The Authorization Server MUST be able to determine the appropriate client_id from the context of the request without requiring the

Client to explicitly send its own "client_id" in the request.

Operations on this endpoint are switched through the use of specific HTTP verbs.

4.1. Client Update Request

This operation updates a previously-registered client with new metadata at the Authorization Server. This request is authenticated by the Registration Access Token issued to the client.

The Client makes an HTTP PUT request to the Client Update Endpoint with a content type of "application/json". This request MAY include any fields described in Client Metadata (Section 2). If included in the request, valid values of Client Metadata fields in this request MUST replace, not augment, the values previously associated with this Client. Any fields with the value of a JSON "null" in Client Metadata MUST be taken as a request to clear any existing value of that field. Omitted values in the Client Metadata MUST remain unchanged by the Authorization Server. The Authorization Server MAY replace any invalid values with suitable values, and it MUST return any such fields to the Client in the response.

For example, a client could send the following request to the Client Registration Endpoint to update the client registration in the above example:

Following is a non-normative example request (with line wraps for display purposes only):

```
PUT /register/s6BhdRkqt3 HTTP/1.1
Accept: application/json
Host: server.example.com
Authorization: Bearer reg-23410913-abewfq.123483

{
 "redirect_uri":["https://client.example.org/callback",
    "https://client.example.org/alt"],
 "client_name":"My New Example",
 "logo_url":"https://client.example.org/newlogo.png"
}
```

4.2. Client Read Request

In order to read the current configuration of the Client on the Authorization Server, the Client makes an HTTP GET request to the Client Update Endpoint with the Registration Access Token.

Following is a non-normative example request (with line wraps for display purposes only):

```
GET /register/s6BhdRkqt3 HTTP/1.1
Accept: application/json
Host: server.example.com
Authorization: Bearer reg-23410913-abewfq.123483
```

4.3. Client Update or Read Response

Upon successful update or read operation, the Client Update Endpoint returns the Client ID. Additionally, the Authorization Server SHOULD return all registered metadata (Section 2) about this client, including any fields provisioned by the Authorization Server itself.

The Authorization Server MAY reject or replace any of the client's requested metadata values submitted during an update request and substitute them with suitable values. If the Authorization Server performs any such substitutions to the requested values, it MUST return these values in the response.

The Authorization Server MUST NOT include the Client Secret or Request Access Token in this response.

The response is a JSON Document [RFC4627] with the following fields as well as any applicable client metadata as top-level members of a JSON object.

client_id
: REQUIRED. The unique Client identifier, MUST equal the value of the client_id returned in the original client_register request.

_links
: REQUIRED. A JSON object that contains references to the Client Update Endpoint and Client Secret Rotation Endpoint, via the following members:

  self REQUIRED. A JSON object that contains the member href which contains the fully qualified URL of the Client Update Endpoint for this client. This MAY be constructed using a URL Template of the Client Registration Endpoint with the issued client_id.

  rotate_secret REQUIRED. A JSON object that contains the member href which contains the fully qualified URL of the Client Secret Rotation Endpoint for this client. This MAY be constructed using a URL Template of the Client Registration Endpoint with the issued client_id.

Following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
 _links: {
   "self": {
     "href": "https://server.example.com/register/s6BhdRkqt3"
   },
   "rotate_secret": {
     "href": "https://server.example.com/register/s6BhdRkqt3/secret"
   }
 "client_id": "s6BhdRkqt3",
 "client_name": "My New Example",
 "redirect_uri": ["https://client.example.org/callback",
   "https://client.example.org/alt"]
 "scope": "read write dolphin",
 "grant_type": ["authorization_code", "refresh_token"],
 "token_endpoint_auth_method": "client_secret_basic",
 "logo_url": "https://client.example.org/newlogo.png",
 "jwk_url": "https://client.example.org/my_rsa_public_key.jwk",
}
```

4.4. Client Delete Request

In order to deprovision itself on the Authorization Server, the Client makes an HTTP DELETE request to the Client Update Endpoint with the Registration Access Token. This request is authenticated by the Registration Access Token issued to the client.

Following is a non-normative example request (with line wraps for display purposes only):

```
DELETE /register/s6BhdRkqt3 HTTP/1.1
Accept: application/json
Host: server.example.com
Authorization: Bearer reg-23410913-abewfq.123483
```

If a client has been successfully deprovisioned, the Authorization Server responds with an HTTP 204 No Content message.

Following is a non-normative example response:

```
HTTP/1.1 204 No Content
Cache-Control: no-store
```

5. Client Secret Rotation

The Client Secret Rotation Endpoint is an OAuth 2.0 protected endpoint that is provisioned by the server for a specific client to be able to request rotation of its Registration Access Token and, if it has one, Client Secret. It is RECOMMENDED that this endpoint URL be formed through the use of a URL template which combines the Client Registration Endpoint and the issued client_id for this client, either as a path parameter (https://server.example.com/register/rotate_secret/client_id) or as a query parameter (https://server.example.com/register/?rotate_secret=client_id). The Authorization Server MUST provide the client with the fully qualified URL in the _links structure described in section 3, and MUST NOT require the Client to construct this URL on its own.

The Authorization Server MUST be able to determine the appropriate client_id from the context of the request without requiring the Client to explicitly send its own "client_id" in the request.

5.1. Rotate Secret Request

This operation allows the client to rotate its current Registration Access Token as well as its Client Secret, if it has one. The client sends an HTTP POST with its current Registration Access Token. This request is authenticated by the Registration Access Token issued to the client.

Following is a non-normative example request (with line wraps for display purposes only):

```
POST /register/rotate_secret/s6BhdRkqt3 HTTP/1.1
Accept: application/json
Host: server.example.com
Authorization: Bearer reg-23410913-abewfq.123483
```

5.2. Rotate Secret Response

Upon successful rotation of the Registration Access Token, and optionally the Client Secret, the Client Registration Endpoint returns a JSON document [RFC4627] with the following fields as top-level members of the root JSON object. This response MUST NOT include any other client metadata.

client_id
: REQUIRED. The unique Client identifier, MUST match the client_id issued in the original registration request.

client_secret
REQUIRED if the server initially issued this Client a Client Secret, otherwise the server MUST NOT return a value. The value MUST be unique for each "client_id".

registration_access_token
REQUIRED. The Access token to be used by the client to perform subsequent "client_update" and "rotate_secret" requests.

issued_at
OPTIONAL. Specifies the timestamp when the identifier was issued. The timestamp value MUST be a positive integer. The value is expressed in the number of seconds since January 1, 1970 00:00:00 GMT.

expires_at
REQUIRED if the server issues a Client Secret. The number of seconds from 1970-01-01T0:0:0Z as measured in UTC that the "client_secret" will expire or "0" if they do not expire. See RFC 3339 [RFC3339] for details regarding date/times in general and UTC in particular.

Following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
 "client_id":"s6BhdRkqt3",
 "client_secret": "7fce6c93f31185e5885805d",
  "registration_access_token": "reg-02348913-oieqer.983421",
 "expires_at":2893276800
}
```

The Authorization Server SHOULD discard and invalidate the Request Access Token and the Client Secret associated with this Client after successful completion of this request.

6. IANA Considerations

This document makes no requests of IANA.

7. Security Considerations

[[Editor's note: Following are some security considerations taken from the UMA and OpenID Connect source drafts. These need to be

massaged into a properly generic set of considerations.]]

Since requests to the Client Registration Endpoint result in the transmission of clear-text credentials (in the HTTP request and response), the server MUST require the use of a transport-layer security mechanism when sending requests to the Registration Endpoint. The server MUST support TLS 1.2 RFC 5246 [RFC5246] and/or TLS 1.0 [RFC2246] and MAY support additional transport-layer mechanisms meeting its security requirements. When using TLS, the Client MUST perform a TLS/SSL server certificate check, per RFC 6125 [RFC6125].

As this endpoint is an OAuth2 Protected Resource, requests to the Registration Endpoint SHOULD have some rate limiting on failures to prevent the Registration Access Token from being disclosed though repeated access attempts.

The authorization server MUST treat all client metadata as self-asserted. A rogue Client might use the name and logo for the legitimate Client, which it is trying to impersonate. An Authorization Server needs to take steps to mitigate this phishing risk, since the logo could confuse users into thinking they're logging in to the legitimate Client. For instance, an Authorization Server could warn if the domain/site of the logo doesn't match the domain/site of redirect URIs. An Authorization Server can also present warning messages to end users about untrusted Clients in all cases, especially if such clients have been dynamically registered and have not been trusted by any users at the Authorization Server before.

In a situation where the Authorization Server is supporting open Client registration, it must be extremely careful with any URL provided by the Client that will be displayed to the user (e.g. "logo_url" and "policy_url"). A rogue Client could specify a registration request with a reference to a drive-by download in the "policy_url". The Authorization Server should check to see if the "logo_url" and "policy_url" have the same host as the hosts defined in the array of "redirect_uris".

While the Client Secret can expire, the Registration Access Token should not expire while a client is still actively registered. If this token were to expire, a Client could be left in a situation where it has no means of updating itself and must register itself anew. As the Registration Access Tokens are long-term credentials, they MUST be protected by the Client as a secret. [[Editor's note: with the right error codes returned from client_update, the AS could force the Client to call rotate_secret before going forward, lessening the window for abuse of a leaked registration token.]]

Since the Registration Access Token is a Bearer token and acts as the sole authentication for use at the Client Update Endpoint, it MUST be protected by the Client as described in OAuth 2.0 Bearer [RFC6750].

8. Acknowledgments

The authors thank the OAuth Working Group, the User-Managed Access Working Group, and the OpenID Connect Working Group participants for their input to this document. In particular, the following individuals have been instrumental in their review and contribution to various versions of this document: Torsten Lodderstedt, Eve Maler, Thomas Hardjono, Christian Scholz, Nat Sakimura, George Fletcher, Amanda Anganes, and Domenico Catalano.

9. Document History

[[to be removed by RFC editor before publication as an RFC]]

- 05

o changed redirect_uri and contact to lists instead of space delimited strings

o removed operation parameter

o added _links structure

o made client update management more RESTful

o split endpoint into three parts

o changed input to JSON from form-encoded

o added READ and DELETE operations

o removed Requirements section

o changed token_endpoint_auth_type back to token_endpoint_auth_method to match OIDC who changed to match us

- 04

o removed default_acr, too undefined in the general OAuth2 case

o removed default_max_auth_age, since there's no mechanism for supplying a non-default max_auth_age in OAuth2

- clarified signing and encryption URLs
- changed token_endpoint_auth_method to token_endpoint_auth_type to match OIDC

- 03

- added scope and grant_type claims
- fixed various typos and changed wording for better clarity
- endpoint now returns the full set of client information
- operations on client_update allow for three actions on metadata: leave existing value, clear existing value, replace existing value with new value

- 02

- Reorganized contributors and references
- Moved OAuth references to RFC
- Reorganized model/protocol sections for clarity
- Changed terminology to "client register" instead of "client associate"
- Specified that client_id must match across all subsequent requests
- Fixed RFC2XML formatting, especially on lists

- 01

- Merged UMA and OpenID Connect registrations into a single document
- Changed to form-paramter inputs to endpoint
- Removed pull-based registration

- 00

- Imported original UMA draft specification

10. Normative References

[JWK] Jones, M., "JSON Web Key (JWK)", May 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012.

Authors' Addresses

Justin Richer (editor)
The MITRE Corporation

Phone:
Fax:
Email: jricher@mitre.org
URI:

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com

Michael B. Jones
Microsoft

Email: mbj@microsoft.com

Maciej Machulak
Newcastle University

Email: m.p.machulak@ncl.ac.uk
URI: http://ncl.ac.uk/