OAuth Working Group J. Richer
Internet-Draft The MITRE Corporation
Intended status: Standards Track M. Jones
Expires: August 1, 2014 Microsoft
J. Bradley
Ping Identity
M. Machulak
Newcastle University
January 28, 2014

OAuth 2.0 Dynamic Client Registration Core Protocol
draft-ietf-oauth-dyn-reg-15

Abstract

This specification defines mechanisms used to dynamically register OAuth 2.0 clients at authorization servers.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

In order for an OAuth 2.0 client to utilize an OAuth 2.0 authorization server, the client needs specific information to interact with the server, including an OAuth 2.0 Client ID to use at that server. This specification describes how an OAuth 2.0 client can be dynamically registered with an authorization server to obtain this information.

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 set of valid redirection URIs. This metadata can either be communicated in a self-asserted fashion or as a set of signed metadata called a software statement; in the case of a software statement, the signer is vouching for the validity of the data about the client.

The mechanisms defined in this specification can be used either for a client to dynamically register itself with authorization servers or for a client developer to programmatically register the client with authorization servers.

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", "Response Type", and "Token Endpoint" defined by OAuth 2.0 [RFC6749] and uses the term "Claim" defined by JSON Web Token (JWT) [JWT].

This specification defines the following terms:

Client Developer The person or organization that builds a client software package and prepares it for distribution. A client developer obtains a software assertion from a software publisher, or self-generates one for the purposes of facilitating client registration.

Client Instance A deployed instance of a piece of client software. Multiple instances of the same piece of client software MAY use the same Client ID value at an authorization server, provided that the Redirection URI values and potentially other values dictated by authorization server policy are the same for all instances.

Client Software Software implementing an OAuth 2.0 client.

Client Registration Endpoint OAuth 2.0 endpoint through which a client can be registered at an authorization server. The means by which the URL for this endpoint is obtained are out of scope for this specification.

Initial Access Token OAuth 2.0 access token optionally issued by an Authorization Server and used to authorize calls to the client registration endpoint. The type and format of this token are likely service-specific and are out of scope for this specification. The means by which the authorization server issues this token as well as the means by which the registration endpoint validates this token are out of scope for this specification.

Deployment Organization An administrative security domain under which, a software API is deployed and protected by an OAuth 2.0 framework. In simple cloud deployments, the software API publisher and the deployment organization may be the same. In other scenarios, a software publisher may be working with many different deployment organizations.

Software API Deployment A deployment instance of a software API that is protected by OAuth 2.0 in a particular deployment organization domain. For any particular software API, there may be one or more deployments. A software API deployment typically has an associated OAuth 2.0 authorization server endpoint as well as a client registration endpoint. The means by which endpoints are obtained (discovery) are out of scope for this specification.

Software API Publisher The organization that defines a particular web accessible API that may deployed in one or more deployment environments. A publisher may be any commercial, public, private, or open source organization that is responsible for publishing and distributing software that may be protected via OAuth 2.0. A software API publisher may issue software assertions which client developers use to distribute with their software to facilitate registration. In some cases a software API publisher and a client developer may be the same organization.

Software Statement A signed JSON Web Token (JWT) [JWT] that asserts metadata values about the client software. This may be used by the registration system to qualify clients for eligibility to register. To obtain a software statement, a client developer may generate a client specific assertion, or a client developer may register with a software API publisher to obtain a software assertion. The statement is distributed with all copies of a client application and may be used during the registration process.

1.3. Protocol Flow

```
     +--------(A)- Initial Access Token (OPTIONAL)
     |
     |   +----(B)- Software Statement (OPTIONAL)
     |   |
     v   v
 +-----------+                                      +---------------+
 |           |--(C)- Client Registration Request -->|    Client     |
 | Client or |                                      | Registration  |
 | Developer |<-(D)- Client Information Response ---|   Endpoint    |
 |           |                                      +---------------+
 +-----------+
```

Figure 1: Abstract Dynamic Client Registration Flow

The abstract OAuth 2.0 client dynamic registration flow illustrated in Figure 1 describes the interaction between the client or developer and the endpoint defined in this specification. This figure does not demonstrate error conditions. This flow includes the following steps:

(A) Optionally, the client or developer is issued an initial access token giving access to the client registration endpoint. The method by which the initial access token is issued to the client or developer is out of scope for this specification.

(B) Optionally, the client or developer is issued a software statement for use with the client registration endpoint. The method by which the software statement is issued to the client or developer is out of scope for this specification.

(C) The client or developer calls the client registration endpoint with its desired registration metadata, optionally including the initial access token from (A) if one is required by the authorization server.

(D) The authorization server registers the client and returns the client's registered metadata, a client identifier that is unique at the server, a set of client credentials such as a client secret if applicable for this client, and possibly other values.

2. Client Metadata

Clients have a set of metadata values associated with their unique client identifier at an authorization server, such as the list of valid redirect URIs.

The Client Metadata values are used in two ways:

o as input values to registration requests, and

o as output values in registration responses.

Client Metadata values can either be communicated directly in the body of a registration request, as described in Section 4.1, or included as claims in a software statement, as described in Section 3.

These Client Metadata values are defined by this specification:

redirect_uris Array of redirect URIs for use in redirect-based flows such as the authorization code and implicit grant types. It is RECOMMENDED that clients using these flows register this parameter, and an authorization server SHOULD require registration of valid redirect URIs for all clients that use these grant types to protect against token and credential theft attacks.

token_endpoint_auth_method The requested authentication method for the token endpoint. Values defined by this specification are:

* "none": The client 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 OAuth 2.0 section 2.3.1.

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

Additional values can be defined via the IANA OAuth Token Endpoint Authentication Methods Registry Section 6.2. Absolute URIs can also be used as values for this parameter without being registered. 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.

grant_types Array of OAuth 2.0 grant types that the Client may use. These grant types are defined as follows:

* "authorization_code": The Authorization Code Grant described in OAuth 2.0 Section 4.1

* "implicit": The Implicit Grant described in OAuth 2.0 Section 4.2

* "password": The Resource Owner Password Credentials Grant described in OAuth 2.0 Section 4.3

* "client_credentials": The Client Credentials Grant described in OAuth 2.0 Section 4.4

* "refresh_token": The Refresh Token Grant described in OAuth 2.0 Section 6.

* "urn:ietf:params:oauth:grant-type:jwt-bearer": The JWT Bearer Grant defined in OAuth JWT Bearer Token Profiles [OAuth.JWT].

* "urn:ietf:params:oauth:grant-type:saml2-bearer": The SAML 2 Bearer Grant defined in OAuth SAML 2 Bearer Token Profiles [OAuth.SAML2].

Authorization Servers MAY allow for other values as defined in grant type extensions to OAuth 2.0. The extension process is described in OAuth 2.0 Section 2.5. If the token endpoint is used in the grant type, the value of this parameter MUST be the same as the value of the "grant_type" parameter passed to the token endpoint defined in the extension.

response_types Array of the OAuth 2.0 response types that the Client may use. These response types are defined as follows:

* "code": The Authorization Code response described in OAuth 2.0 Section 4.1.

* "token": The Implicit response described in OAuth 2.0 Section 4.2.

Authorization servers MAY allow for other values as defined in response type extensions to OAuth 2.0. The extension process is described in OAuth 2.0 Section 2.5. If the authorization endpoint is used by the grant type, the value of this parameter MUST be the

same as the value of the "response_type" parameter passed to the authorization endpoint defined in the extension.

Authorization servers MUST accept all fields in this list. Extensions and profiles of this specification MAY expand this list. For instance, the [OAuth.Registration.Metadata] specification defines additional client metadata values. The authorization server MUST ignore any client metadata values sent by the Client that it does not understand.

2.1. Relationship between Grant Types and Response Types

The "grant_types" and "response_types" values described above are partially orthogonal, as they refer to arguments passed to different endpoints in the OAuth protocol. However, they are related in that the "grant_types" available to a client influence the "response_types" that the client is allowed to use, and vice versa. For instance, a "grant_types" value that includes "authorization_code" implies a "response_types" value that includes "code", as both values are defined as part of the OAuth 2.0 authorization code grant. As such, a server supporting these fields SHOULD take steps to ensure that a client cannot register itself into an inconsistent state.

The correlation between the two fields is listed in the table below.

grant_types value includes:	response_types value includes:
authorization_code	code
implicit	token
password	(none)
client_credentials	(none)
refresh_token	(none)
urn:ietf:params:oauth:grant-type:jwt-bearer	(none)
urn:ietf:params:oauth:grant-type:saml2-bearer	(none)

Extensions and profiles of this document that introduce new values to either the "grant_types" or "response_types" parameter MUST document all correspondences between these two parameter types.

3. Software Statement

A Software Statement is a signed JSON Web Token (JWT) [JWT] that asserts metadata values about the client software. This may be used

by the registration system to qualify clients for eligibility to register. To obtain a software statement, a client developer may generate a client specific assertion, or a client developer may register with a software API publisher to obtain a software assertion. The statement is distributed with all copies of a client application and may be used during the registration process.

The criteria by which authorization servers determine whether to trust and utilize the information in a software statement is beyond the scope of this specification.

If the authorization server determines that the claims in a software statement uniquely identify a piece of software, the same Client ID value MAY be returned for all dynamic registrations using that software statement.

In some cases, authorization servers MAY choose to accept a software statement value directly as a Client ID in an authorization request, without a prior dynamic client registration having been performed. The circumstances under which an authorization server would do so, and the specific software statement characteristics required in this case, are beyond the scope of this specification.

4. 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 be registered 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 be an OAuth 2.0 protected resource and accept an initial access token in the form of an OAuth 2.0 [RFC6749] access token to limit registration to only previously authorized parties. The method by which the initial access token is obtained by the registrant is generally out-of-band and is out of scope for this specification. The method by which the initial access token is verified and validated by the client registration endpoint is out of scope for this specification.

To support open registration and facilitate wider interoperability,

the client registration endpoint SHOULD allow initial registration requests with no authorization (which is to say, with no OAuth 2.0 access token in the request). These requests MAY be rate-limited or otherwise limited to prevent a denial-of-service attack on the client registration endpoint.

The client registration endpoint MUST ignore all parameters it does not understand.

4.1. 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 client metadata parameters being specified for the client during the registration. The authorization server MAY provision default values for any items omitted in the client metadata.

Client metadata values may also be provided in a software statement, as described in Section 3. Software statements are included in registration requests using this registration parameter:

software_statement A software statement containing client metadata values about the client software as claims.

To register, the client or developer sends an HTTP POST to the client registration endpoint with a content type of "application/json". The HTTP Entity Payload is a JSON [RFC4627] document consisting of a JSON object and all parameters as top-level members of that JSON object.

For example, if the server supports open registration (with no initial access token), the client could send the following registration request to the client registration endpoint:

The following is a non-normative example request not using an initial access token (with line wraps within values for display purposes only):

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

{
 "redirect_uris":[
   "https://client.example.org/callback",
   "https://client.example.org/callback2"],
 "token_endpoint_auth_method":"client_secret_basic",
 "example_extension_parameter": "example_value"
}
```

Alternatively, if the server supports authorized registration, the developer or the client will be provisioned with an initial access token (the method by which the initial access token is obtained is out of scope for this specification). The developer or client sends the following authorized registration request to the client registration endpoint. Note that the initial access token sent in this example as an OAuth 2.0 Bearer Token [RFC6750], but any OAuth 2.0 token type could be used by an authorization server.

The following is a non-normative example request using an initial access token (with line wraps within values for display purposes only):

```
POST /register HTTP/1.1
Content-Type: application/json
Accept: application/json
Authorization: Bearer ey23f2.adfj230.af32-developer321
Host: server.example.com

{
 "redirect_uris":["https://client.example.org/callback",
    "https://client.example.org/callback2"],
 "token_endpoint_auth_method":"client_secret_basic",
 "example_extension_parameter": "example_value"
}
```

In the following example, some registration parameters are conveyed as claims in a software statement (with line wraps within values for display purposes only):

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

{
  "redirect_uris":[
    "https://client.example.org/callback",
    "https://client.example.org/callback2"
  ],
  "software_statement":"eyJhbGciOiJFUzI1NiJ9.
     eyJpc3Mi[...omitted for brevity...].
     J9l-ZhwP[...omitted for brevity...]",
  "extension_parameter":"foo"
}
```

4.2. Client Registration Response

Upon successful registration, the authorization server generates a new client identifier for the client. This client identifier MUST be unique at the server and MUST NOT be in use by any other client. The server responds with an HTTP 201 Created code and a body of type "application/json" with content as described in Section 5.1.

Upon an unsuccessful registration, the authorization server responds with an error, as described in Section 5.2.

5. Responses

In response to certain requests from the client to either the client registration endpoint as described in this specification, the authorization server sends the following response bodies.

5.1. Client Information Response

The response contains the client identifier as well as the client secret, if the client is a confidential client. The response MAY contain additional fields as specified by extensions to this specification.

client_id REQUIRED. Unique client identifier. It MUST NOT be currently valid for any other distinct registered client. It MAY be the same as the Client ID value used by other instances of this client, provided that the Redirection URI values and potentially other values dictated by authorization server policy are the same for all instances.

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 [RFC6749] Section 2.3.1.

client_id_issued_at OPTIONAL. Time at which the Client Identifier was issued. The time is represented as the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

client_secret_expires_at REQUIRED if "client_secret" is issued. Time at which the "client_secret" will expire or 0 if it will not expire. The time is represented as the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

Additionally, the Authorization Server MUST return all registered metadata 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 or update requests and substitute them with suitable values.

The response is an "application/json" document with all parameters as top-level members of a JSON object [RFC4627].

If a software statement was used as part of the registration, its value SHOULD be returned in the response. Client metadata elements used from the software statement SHOULD also be returned directly as top-level client metadata values in the registration response.

Following is a non-normative example response:

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

{
 "client_id":"s6BhdRkqt3",
 "client_secret": "cf136dc3c1fc93f31185e5885805d",
 "client_id_issued_at":2893256800,
 "client_secret_expires_at":2893276800,
 "redirect_uris":[
   "https://client.example.org/callback",
   "https://client.example.org/callback2"],
 "grant_types": ["authorization_code", "refresh_token"],
 "token_endpoint_auth_method": "client_secret_basic",
 "example_extension_parameter": "example_value"
}
```

5.2. Client Registration Error Response

When an OAuth 2.0 error condition occurs, such as the client presenting an invalid initial access token, the authorization server returns an error response appropriate to the OAuth 2.0 token type.

When a registration error condition occurs, the authorization server returns an HTTP 400 status code (unless otherwise specified) with content type "application/json" consisting of a JSON object [RFC4627] describing the error in the response body.

The JSON object contains two members:

error Single ASCII error code string.

error_description Human-readable ASCII text description of the error used 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 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.

invalid_software_statement The software statement presented is invalid.

unapproved_software_statement The software statement presented is not approved for use with this authorization server.

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
Pragma: no-cache

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

6. IANA Considerations

6.1. OAuth Registration Client Metadata Registry

This specification establishes the OAuth Registration Client Metadata registry.

OAuth registration client metadata values are registered with a Specification Required ([RFC5226]) after a two-week review period on the oauth-ext-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests must be sent to the oauth-ext-review@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request to register OAuth Registration Client Metadata name: example").

Within the review period, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

IANA must only accept registry updates from the Designated Expert(s)

and should direct all requests for registration to the review mailing list.

6.1.1. Registration Template

Client Metadata Name: The name requested (e.g., "example"). This name is case sensitive. Names that match other registered names in a case insensitive manner SHOULD NOT be accepted.

Client Metadata Description:
Brief description of the metadata value (e.g., "Example description").

Change controller: For Standards Track RFCs, state "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification document(s): Reference to the document(s) that specify the token endpoint authorization method, preferably including a URI that can be used to retrieve a copy of the document(s). An indication of the relevant sections may also be included but is not required.

6.1.2. Initial Registry Contents

The initial contents of the OAuth Registration Client Metadata registry are:

o Client Metadata Name: "redirect_uris"
o Client Metadata Description: Array of redirect URIs for use in redirect-based flows
o Change controller: IESG
o Specification document(s): [[this document]]

o Client Metadata Name: "token_endpoint_auth_method"
o Client Metadata Description: Requested authentication method for the token endpoint
o Change controller: IESG
o Specification document(s): [[this document]]

o Client Metadata Name: "grant_types"
o Client Metadata Description: Array of OAuth 2.0 grant types that the Client may use
o Change controller: IESG
o Specification document(s): [[this document]]

- Client Metadata Name: "response_types"
- Client Metadata Description: Array of the OAuth 2.0 response types that the Client may use
- Change controller: IESG
- Specification document(s): [[this document]]

6.2. OAuth Token Endpoint Authentication Methods Registry

This specification establishes the OAuth Token Endpoint Authentication Methods registry.

Additional values for use as "token_endpoint_auth_method" metadata values are registered with a Specification Required ([RFC5226]) after a two-week review period on the oauth-ext-review@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests must be sent to the oauth-ext-review@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request to register token_endpoint_auth_method value: example").

Within the review period, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

IANA must only accept registry updates from the Designated Expert(s) and should direct all requests for registration to the review mailing list.

6.2.1. Registration Template

Token Endpoint Authorization Method Name: The name requested (e.g., "example"). This name is case sensitive. Names that match other registered names in a case insensitive manner SHOULD NOT be accepted.

Change controller: For Standards Track RFCs, state "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification document(s): Reference to the document(s) that specify the token endpoint authorization method, preferably including a URI that can be used to retrieve a copy of the document(s). An indication of the relevant sections may also be included but is not required.

6.2.2. Initial Registry Contents

The initial contents of the OAuth Token Endpoint Authentication Methods registry are:

- Token Endpoint Authorization Method Name: "none"
- Change controller: IESG
- Specification document(s): [[this document]]

- Token Endpoint Authorization Method Name: "client_secret_post"
- Change controller: IESG
- Specification document(s): [[this document]]

- Token Endpoint Authorization Method Name: "client_secret_basic"
- Change controller: IESG
- Specification document(s): [[this document]]

7. Security Considerations

Since requests to the client registration endpoint result in the transmission of clear-text credentials (in the HTTP request and response), the Authorization 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].

For clients that use redirect-based grant types such as "authorization_code" and "implicit", authorization servers SHOULD require clients to register their "redirect_uris". Requiring clients to do so can help mitigate attacks where rogue actors inject and impersonate a validly registered client and intercept its authorization code or tokens through an invalid redirect URI.

Public clients MAY register with an authorization server using this protocol, if the authorization server's policy allows them. Public clients use a "none" value for the "token_endpoint_auth_method" metadata field and are generally used with the "implicit" grant type. Often these clients will be short-lived in-browser applications

requesting access to a user's resources and access is tied to a user's active session at the authorization server. Since such clients often do not have long-term storage, it's possible that such clients would need to re-register every time the browser application is loaded. Additionally, such clients may not have ample opportunity to unregister themselves using the delete action before the browser closes. To avoid the resulting proliferation of dead client identifiers, an authorization server MAY decide to expire registrations for existing clients meeting certain criteria after a period of time has elapsed.

Since different OAuth 2.0 grant types have different security and usage parameters, an authorization server MAY require separate registrations for a piece of software to support multiple grant types. For instance, an authorization server might require that all clients using the "authorization_code" grant type make use of a client secret for the "token_endpoint_auth_method", but any clients using the "implicit" grant type do not use any authentication at the token endpoint. In such a situation, a server MAY disallow clients from registering for both the "authorization_code" and "implicit" grant types simultaneously. Similarly, the "authorization_code" grant type is used to represent access on behalf of an end user, but the "client_credentials" grant type represents access on behalf of the client itself. For security reasons, an authorization server could require that different scopes be used for these different use cases, and as a consequence it MAY disallow these two grant types from being registered together by the same client. In all of these cases, the authorization server would respond with an "invalid_client_metadata" error response.

8. References

8.1. Normative References

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token (work in progress), January 2014.

[OAuth.JWT] Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", draft-ietf-oauth-jwt-bearer (work in progress), December 2013.

[OAuth.SAML2] Campbell, B., Mortimore, C., and M. Jones, "SAML 2.0 Profile for OAuth 2.0 Client Authentication and

Authorization Grants", draft-ietf-oauth-saml2-bearer (work in progress), December 2013.

[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.

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[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.

8.2. Informative References

[OAuth.Registration.Management]
Richer, J., Jones, M., Bradley, J., and M. Machulak, "OAuth 2.0 Dynamic Client Registration Management Protocol", draft-jones-oauth-dyn-reg-management (work in progress), January 2014.

[OAuth.Registration.Metadata]
Richer, J., Jones, M., Bradley, J., and M. Machulak, "OAuth 2.0 Dynamic Client Registration Metadata", draft-jones-oauth-dyn-reg-metadata (work in progress), January 2014.

Appendix A. Use Cases

This appendix describes different ways that this specification can be utilized, including describing some of the choices that may need to be made. Some of the choices are independent and can be used in combination, whereas some of the choices are interrelated.

A.1. Open versus Protected Dynamic Client Registration

A.1.1. Open Dynamic Client Registration

Authorization servers that support open registration allow registrations to be made with no initial access token. This allows all client software to register with the authorization server.

A.1.2. Protected Dynamic Client Registration

Authorization servers that support protected registration require that an initial access token be used when making registration requests. While the method by which a client or developer receives this initial access token and the method by which the authorization server validates this initial access token are out of scope for this specification, a common approach is for the developer to use a manual pre-registration portal at the authorization server that issues an initial access token to the developer.

A.2. Registration without or with Software Statements

A.2.1. Registration without a Software Statement

When a software statement is not used in the registration request, the authorization server must be willing to use client metadata values without them being signed (and thereby attested to) by any authority. (Note that this choice is independent of the Open versus Protected choice, and that an initial access token is another possible form of attestation.)

A.2.2. Registration with a Software Statement

A software statement can be used in a registration request to provide attestation for a set of client metadata values for a piece of client software by an authority. This can be useful when the authorization server wants to restrict registration to client software attested to by a set of authorities or when it wants to know that multiple registration requests refer to the same piece of client software.

A.3. Registration by the Client or the Developer

A.3.1. Registration by the Client

In some use cases, client software will dynamically register itself with an authorization server to obtain a Client ID and other information needed to interact with the authorization server. In this case, no Client ID for the authorization server is packaged with the client software.

A.3.2. Registration by the Developer

In some cases, the developer (or development software being used by the developer) will pre-register the client software with the authorization server or a set of authorization servers. In this case, the Client ID value(s) for the authorization server(s) can be packaged with the client software.

A.4. Client ID per Client Instance or per Client Software

A.4.1. Client ID per Client Software Instance

In some cases, each deployed instance of a piece of client software will dynamically register and obtain distinct Client ID values. This can be advantageous, for instance, if the code flow is being used, as it also enables each client instance to have its own client secret. This can be useful for native clients, which cannot maintain the secrecy of a client secret value packaged with the software, but which may be able to maintain the secrecy of a per-instance client secret.

A.4.2. Client ID Shared between all Instances of Client Software

In some cases, each deployed instance of a piece of client software will share a common Client ID value. For instance, this is often the case for native client using implicit flow, when no client secret is involved. Particular authorization servers might choose, for instance, to maintain a mapping between software statement values and Client ID values, and return the same Client ID value for all registration requests for a particular piece of software. The circumstances under which an authorization server would do so, and the specific software statement characteristics required in this case, are beyond the scope of this specification.

A.5. Stateful or Stateless Registration

A.5.1. Stateful Client Registration

In some cases, authorization servers will maintain state about registered clients, typically indexing this state using the Client ID value. This state would typically include the client metadata values associated with the client registration, and possibly other state specific to the authorization server's implementation. When stateful registration is used, operations to support retrieving and/or updating this state may be supported, as described in the [OAuth.Registration.Management] specification.

A.5.2. Stateless Client Registration

In some cases, authorization servers will be implemented in a manner the enables them to not maintain any local state about registered clients. One means of doing this is to encode all the registration state in the returned Client ID value, and possibly encrypting the state to the authorization server to maintain the confidentiality and integrity of the state.

Appendix B. 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: Amanda Anganes, Derek Atkins, Tim Bray, Domenico Catalano, Donald Coffin, Vladimir Dzhuvinov, George Fletcher, Thomas Hardjono, Phil Hunt, William Kim, Torsten Lodderstedt, Eve Maler, Josh Mandel, Nov Matake, Tony Nadalin, Nat Sakimura, Christian Scholz, and Hannes Tschofenig.

Appendix C. Document History

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

-15

- o Partitioned the Dynamic Client Registration specification into core, metadata, and management specifications. This built on work first published as draft-richer-oauth-dyn-reg-core-00 and draft-richer-oauth-dyn-reg-management-00.

- o Added the ability to use Software Statements. This built on work first published as draft-hunt-oauth-software-statement-00 and draft-hunt-oauth-client-association-00.

- Created the IANA OAuth Registration Client Metadata registry for registering Client Metadata values.

- Defined Client Instance term and stated that multiple instances can use the same Client ID value under certain circumstances.

- Rewrote the introduction.

- Rewrote the Use Cases appendix.

-14

- Added software_id and software_version metadata fields

- Added direct references to RFC6750 errors in read/update/delete methods

-13

- Fixed broken example text in registration request and in delete request

- Added security discussion of separating clients of different grant types

- Fixed error reference to point to RFC6750 instead of RFC6749

- Clarified that servers must respond to all requests to configuration endpoint, even if it's just an error code

- Lowercased all Terms to conform to style used in RFC6750

-12

- Improved definition of Initial Access Token

- Changed developer registration scenario to have the Initial Access Token gotten through a normal OAuth 2.0 flow

- Moved non-normative client lifecycle examples to appendix

- Marked differentiating between auth servers as out of scope

- Added protocol flow diagram

- Added credential rotation discussion

- Called out Client Registration Endpoint as an OAuth 2.0 Protected Resource

- Cleaned up several pieces of text

-11

- Added localized text to registration request and response examples.

- Removed "client_secret_jwt" and "private_key_jwt".

- Clarified "tos_uri" and "policy_uri" definitions.

- Added the OAuth Token Endpoint Authentication Methods registry for registering "token_endpoint_auth_method" metadata values.

- Removed uses of non-ASCII characters, per RFC formatting rules.

- Changed "expires_at" to "client_secret_expires_at" and "issued_at" to "client_id_issued_at" for greater clarity.

- Added explanatory text for different credentials (Initial Access Token, Registration Access Token, Client Credentials) and what they're used for.

- Added Client Lifecycle discussion and examples.

- Defined Initial Access Token in Terminology section.

-10

- Added language to point out that scope values are service-specific

- Clarified normative language around client metadata

- Added extensibility to token_endpoint_auth_method using absolute URIs

- Added security consideration about registering redirect URIs

- Changed erroneous 403 responses to 401's with notes about token handling

- Added example for initial registration credential

-09

- Added method of internationalization for Client Metadata values

- Fixed SAML reference

-08

- Collapsed jwk_uri, jwk_encryption_uri, x509_uri, and x509_encryption_uri into a single jwks_uri parameter

- Renamed grant_type to grant_types since it's a plural value

- Formalized name of "OAuth 2.0" throughout document

- Added JWT Bearer Assertion and SAML 2 Bearer Assertion to example grant types

- Added response_types parameter and explanatory text on its use with and relationship to grant_types

-07

- Changed registration_access_url to registration_client_uri

- Fixed missing text in 5.1

- Added Pragma: no-cache to examples

- Changed "no such client" error to 403

- Renamed Client Registration Access Endpoint to Client Configuration Endpoint

- Changed all the parameter names containing "_url" to instead use "_uri"

- Updated example text for forming Client Configuration Endpoint URL

-06

- Removed secret_rotation as a client-initiated action, including removing client secret rotation endpoint and parameters.

- Changed _links structure to single value registration_access_url.

- Collapsed create/update/read responses into client info response.

- Changed return code of create action to 201.

- Added section to describe suggested generation and composition of Client Registration Access URL.

- Added clarifying text to PUT and POST requests to specify JSON in the body.

- Added Editor's Note to DELETE operation about its inclusion.

- Added Editor's Note to registration_access_url about alternate syntax proposals.

-05

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

- removed operation parameter

- added _links structure

- made client update management more RESTful

- split endpoint into three parts

- changed input to JSON from form-encoded

- added READ and DELETE operations

- removed Requirements section

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

-04

- removed default_acr, too undefined in the general OAuth2 case

- 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-parameter inputs to endpoint
- Removed pull-based registration

-00

- Imported original UMA draft specification

Authors' Addresses

Justin Richer
The MITRE Corporation

Email: jricher@mitre.org

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com

Maciej Machulak
Newcastle University

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