Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 14, 2013

J. Richer, Ed.
The MITRE Corporation
J. Bradley
Ping Identity
M. Jones
Microsoft
M. Machulak
Newcastle University
December 11, 2012

OAuth Dynamic Client Registration Protocol
draft-ietf-oauth-dyn-reg-03

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 June 14, 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 and manage the metadata associated with it.

- 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.

1.3. Requirements

[[Following are proposed requirements for dynamic client registration. This section is intended for discussion and will likely be removed in the final draft.]]

1.3.1. The client needs to be uniquely identifiable by the authorization server

In order for an authorization server to do proper user-delegated authorization and prevent unauthorized access it must be able to identify clients uniquely. As is done today in OAuth, the client identifier (and optional secret) should thus be issued by the authorization server and not simply accepted as proposed by the client.

1.3.2. The authorization server must collect metadata about a client for later user interaction

In order for the authorization server to describe a client to an end-user in an authorization step it needs information about the client. This can be the client name at a minimum, but today servers usually request at least a description, a homepage URL, and an icon when doing manual registration.

1.3.3. The authorization server should have the option of strongly authenticating the client and its metadata

In order to prevent spoofing of clients and enable dynamic building of strong trust relationships, the authorization server should have the option to verify the provided information. This might be solved using message signature verification.

1.3.4. Dynamic client registration must be possible from both web-server applications and applications with other capabilities and limitations, such as native applications

Each instance of a native application (that is, the specific instance running on each device) that is installed and run by the same user may need the option of getting a unique client identifier. In this case, there are implications around gathering and displaying enough information to ensure that the end-user is delegating authorization to the intended application. The registration protocol should be simple and flexible enough to allow for multiple types of applications.

1.3.5. Transaction integrity must be ensured

When a client sends information to a server endpoint, it might take time for this data to propagate through big server installations that spread across various data centers. Care needs to be taken that subsequent interactions with the user after the registration process, such as an authorization request, show the correct data.

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.

redirect_uris
: RECOMMENDED. A space-delimited 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 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 application. If present, the server SHOULD display this image to the end user during approval.

contacts
: OPTIONAL. Space delimited list of email addresses for people responsible for this client. The Authorization Server MAY may these addresses available to end users for support queries. 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" HTTP Basic Authentication Scheme as specified in Section 2.3.1 of OAuth 2.0 [RFC6749].

scope
: OPTIONAL. Space separated list of scopes that the client is allowed to request tokens for. If omitted, an Authorization Server MAY register a Client with a default set of allowed scopes.

grant_type
: OPTIONAL. Space separated 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 Token Endpoint Requests. If jwk_encryption_url is not provided, the key at jwk_url is also used as the key to encrypt responses to the Client. 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 is used to encrypt any 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 Token Endpoint Requests. If "x509_encryption_url" is not provided, "x509_url" it is also used to encrypt responses to the Client. 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 is used to encrypt the ID Token and User Info Endpoint Responses to the Client. If the Client registers both "jwk_encryption_url" and "x509_encryption_url", the keys contained in both formats SHOULD be the same.

default_max_age
: OPTIONAL. Maximum age of a session in integer seconds. Specifies that the End-User must be actively authenticated if any present authentication is older than the specified number of seconds by default.

default_acr
: OPTIONAL. Default Authentication Context class Reference. String that specifies the default authentication context value that the Authorization server must use for processing requests from this client.

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/x-www-form-urlencoded" format. The Client Registration Endpoint MUST be protected by 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].

The Endpoint defines three operations that a client can take on it, switched by the "operation" parameter:

- "client_register": request that the Authorization Server generate a new Client Identifier (and optionally a Client Secret) and associate it with the set of presented metadata (Section 2)

- "client_update": update the metadata (Section 2) associated with a Client Identifier

- "rotate_secret": issue a new Registration Access Token and, if applicable, a Client Secret for a Client

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 "client_register" 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. As such, the Endpoint MUST accept requests with OAuth 2.0 Bearer Tokens [RFC6750] for these operations, whether or not the initial "client_register" call requires authentication of some form.

These two aspects, operation selection and client authentication, are represented by two parameters common to all operations:

operation
: REQUIRED. Values are:

  * "client_register": Register a new client, receive a Client Identifier, and associate metadata with it.

  * "rotate_secret": Request rotation of the Registration Access Token and, if applicable, the Client Secret

  * "client_update": Update the metadata associated with a given Client Identifier.

access_token
: OPTIONAL. An OAuth2 Bearer token used to access the Client Registration Endpoint, as defined in OAuth2 Bearer. This parameter MUST NOT be sent if the Access Token is sent in the HTTP Authorization header as described in Section 7.1 of OAuth 2.0 [RFC6749]. Access Tokens sent in the authorization header must be OAuth 2.0 Bearer Tokens [RFC6750].

Each operation takes a different parameter set, and all operations are described below.

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 the two parameters described above as well as any

parameters described in Client Metadata (Section 2). The Authorization Server MAY provision default values for any items omitted in the Client Metadata.

operation
REQUIRED. MUST have the value "client_register"
access_token
OPTIONAL. Used to restrict new client registration. This parameter MUST NOT be sent if the Access Token is sent in the HTTP Authorization header as described in Section 7.1 of OAuth 2.0 [RFC6749]. Access Tokens sent in the authorization header must be OAuth 2.0 Bearer Tokens [RFC6750].
redirect_uris RECOMMENDED
client_name RECOMMENDED
client_url RECOMMENDED
logo_url OPTIONAL
contacts OPTIONAL
tos_url OPTIONAL
token_endpoint_auth_method OPTIONAL
policy_url OPTIONAL
scope OPTIONAL
grant_type OPTIONAL
jwk_url OPTIONAL
jwk_encryption_url OPTIONAL
x509_url OPTIONAL
x509_encryption_url OPTIONAL
default_max_age OPTIONAL
default_acr OPTIONAL
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/x-www-form-urlencoded
Host: server.example.com

operation=client_register
&redirect_uris=https://client.example.org/callback
    %20https://client.example.org/callback2
&client_name=My%20Example%20Client
&token_endpoint_auth_method=client_secret_basic%20client_secret_post
&scope=read%20write%20dolphin
&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, along with 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 field values and substitute them with suitable values. If this happens, the Authorization Server MUST include these fields in the response to the client.

The response also contains a Registration Access Token that is to be used by the client to perform subsequent operations at this endpoint, such as "client_update" and "rotate_secret".

All of the response items are returned as a JSON document with the following fields as top-level members of the root JSON object.

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

client_secret
: OPTIONAL. The Client secret. 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 "client_update" and "rotate_secret" requests.

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
: OPTIONAL. 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": "cf136dc3c1fc93f31185e5885805d",
 "scope": "read write dolphin",
 "grant_type": "authorization_code refresh_token",
 "token_endpoint_auth_method": "client_secret_basic client_secret_post",
 "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 Update Request

This operation updates a previously-registered client with new metadata at the Authorization Server. This request MUST be protected by the Registration Authorization Token associated with the Client. 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. Empty values 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.

operation
: REQUIRED, MUST have the value "client_update"

access_token
: REQUIRED, unless presented in the Authorization Header as in OAuth2 Bearer [RFC6750]. The Registration Access Token that was issued during the "client_register" step, or previous "client_update" or "rotate_secret" calls.

redirect_uris RECOMMENDED

client_name RECOMMENDED

client_url RECOMMENDED

logo_url OPTIONAL

contacts OPTIONAL

tos_url OPTIONAL

token_endpoint_auth_method OPTIONAL
policy_url OPTIONAL
jwk_url OPTIONAL
jwk_encryption_url OPTIONAL
x509_url OPTIONAL
x509_encryption_url OPTIONAL
default_max_age OPTIONAL
default_acr OPTIONAL
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):

```
POST /register HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: server.example.com
Authorization: Bearer reg-23410913-abewfq.123483

operation=client_update
&redirect_uri=https://client.example.org/callback
    %20https://client.example.org/alt
&client_name=My%20New%20Example%20
&logo_url=https://client.example.org/newlogo.png
```

3.4. Client Update Response

Upon successful update, the Client Registration Endpoint returns the Client ID, along with all current 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 field values and substitute them suitable values. If this happens, the Authorization Server MUST include these fields in the response to the client.

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

These fields are returned as top-level members of the root JSON object.

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

Following is a non-normative example response:

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

{
 "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 client_secret_post",
 "logo_url": "https://client.example.org/newlogo.png",
 "jwk_url": "https://client.example.org/my_rsa_public_key.jwk",
}
```

3.5. 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.

operation
: REQUIRED. MUST have the value rotate_secret

access_token
: REQUIRED, unless presented in the Authorization Header as in OAuth2 Bearer [RFC6750]. The Registration Access Token that was issued during the "client_register" step, or previous "client_update" or "rotate_secret" calls.

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

```
POST /register HTTP/1.1
Accept: application/x-www-form-urlencoded
Host: server.example.com
Authorization: Bearer reg-23410913-abewfq.123483

operation=rotate_secret
```

3.6. Rotate Secret Response

Upon successful rotation of the Registration Access Token, and optionally the Client Secret, the Client Registration Endpoint returns a JSON document 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 client_register 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
OPTIONAL. 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.

3.7. 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 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_operation
: The value of "operation" is invalid or not supported.

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_operation",
 "error_description":"The value of the operation parameter must
  be one of client_register, rotate_secret or client_update."
}
```

4. IANA Considerations

This document makes no requests of IANA.

5. 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.]]

6. 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.

7. Document History

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

- 02

o Reorganized contributors and references

o Moved OAuth references to RFC

o Reorganized model/protocol sections for clarity

o Changed terminology to "client register" instead of "client associate"

o Specified that client_id must match across all subsequent requests

o Fixed RFC2XML formatting, especially on lists

- 01

o Merged UMA and OpenID Connect registrations into a single document

o Changed to form-paramter inputs to endpoint

o Removed pull-based registration

- 00

o Imported original UMA draft specification

8. Normative References

[JWA] Jones, M., "JSON Web Algorithms", May 2012.

[JWE] Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web

Encryption (JWE)", May 2012.

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

[JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature", May 2012.

[JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token", 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.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 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.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[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/