OAuth
Internet-Draft
Intended status: Standards Track
Expires: March 31, 2014

P. Hunt, Ed.
Oracle Corporation
T. Nadalin
Microsoft

September 27, 2013

OAuth Client Association
draft-hunt-oauth-client-association-00

Abstract

This specification defines methods that OAuth clients may use to associate (register) with service providers for the purposes of accessing OAuth protected resources. The document describes different classifications of OAuth clients and the process to directly access or associate for access with a particular OAuth Framework protected service provider.

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 March 31, 2014.

Table of Contents

1. Introduction

The OAuth 2.0 Authorization Framework [RFC6749] is a framework by which client applications use access tokens issued by authorization servers to access to a service provider's software API endpoints. As a framework, OAuth 2.0 enables many different flows by which a client application may obtain an access token including delegated authorization from a user.

The OAuth Authorization Framework defines only two types of clients: public and confidential. Public clients have client_id's issued once where each instance shares the same client_id and are usually native applications. Confidential clients typically have a unique client_id per instance and typically deployed in secure environments on web application platforms. In both cases, OAuth has limited support for building applications that are intended to work with multiple deployments that are not known at compilation or software packaging time.

This specification defines a taxonomy of clients, and the methods by which a client instance may either register with, or directly request tokens from, an OAuth endpoint. The generic term for how client instances work with a new OAuth endpoint is "association". This specification defines 3 types of association:

Static Are clients that are built to work with one or more endpoint(s) that are known at the time the client applicaiton is built. A "client_id" and any associated credentials are typically issued to the developer. Multiple instances of the same client share the same client_id. The determination for "public" vs. "confidential" client is as per Section 2.1 [RFC6749].

Dynamic Are clients that associate with one or more endpoints triggered by application based workflows, configuration or installation events. Associations may be temporary or be extended over a long period of time. A "client_id" is issued at association time along with a token based client credential and an optional client referesh token that enables registration updates and client token rotation. Clients that associate dynamically and are issued individual "client_id" are considered "confidential" as defined in Section 2.1 [RFC6749].

Transient Are clients that associate with one or more endpoints triggered by application based events or workflows. These clients typically use the OAuth "Implicit" grant per Section 4.2 of [RFC6749] and as such do not require an instance specific "client_id" or a client credential. These associations typically exist for the life of an access token and may only last for seconds or minutes. These clients use a client asserted client_id and are considered public as defined in Section 2.1 [RFC6749].

This draft defines how software statements

[I-D.draft-hunt-oauth-software-statement] can be used to associate dynamic and transient clients with OAuth protected service providers.

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", "Public Client", "Confidential Client", "Client Identifier", "Client Secret", "Protected Resource", "Resource Owner", "Resource Server", and "Token Endpoint" defined by OAuth 2.0 [RFC6749].

This specification uses the terms "Deployment Organization", "Software API Deployment", "Software API Publisher", "Client Developer", and "Software Statement" as defined in [I-D.draft-hunt-oauth-software-statement].

This specification defines the following additional terms:

Client Resource Endpoint An optional OAuth 2.0 protected resource endpoint through which registration information for a registered client can be accessed and optionally managed. The API definition is out of scope of this specification.

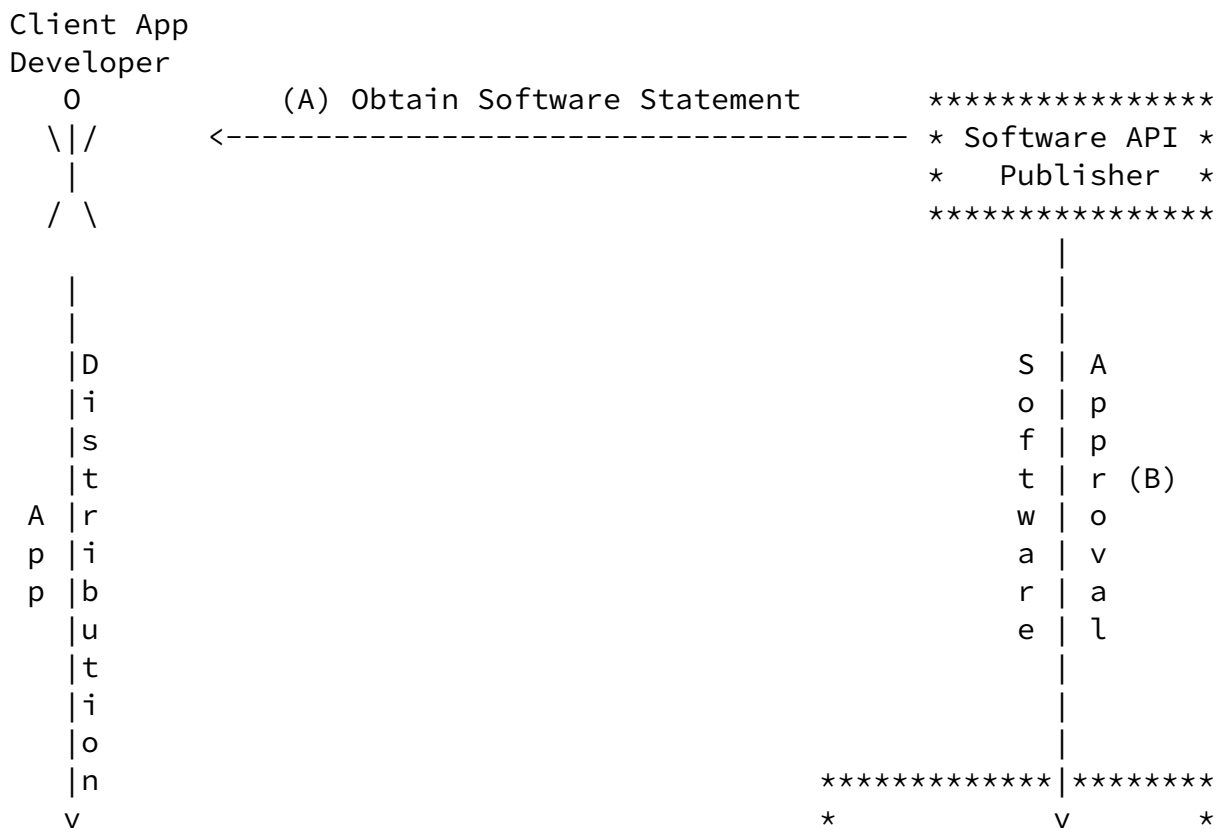
Initial Access Token An OAuth 2.0 access token is typically issued by a software API deployment's security domain and used by a dynamic client a to associate a client for use with a particular software API deployment. The token is usually issued by the same security domain as the Service API the client is registering for. The content, structure, generation, and validation of this token are out of scope for this specification.

Client Refresh Credential A client refresh token is an optional credential token a client may use for the purpose of supporting server or client initiated rotation of client credentials. If client credentials are revoked or expired, the registered client may use the client refresh token to refresh its registration and obtain new client credentials.

2. Client Association Lifecycle

This specification defines an association lifecycle that registers a client for one target resource API per "association". Clients that need to register for more than one resource API should typically make a separate registration request for each API being registered.

The abstract association flow illustrated in Figure 1 describes the relationship and interaction between a software API publisher, a client developer, a deployed client software instance and the software API deployment registration services in this specification. This figure does not demonstrate error conditions.



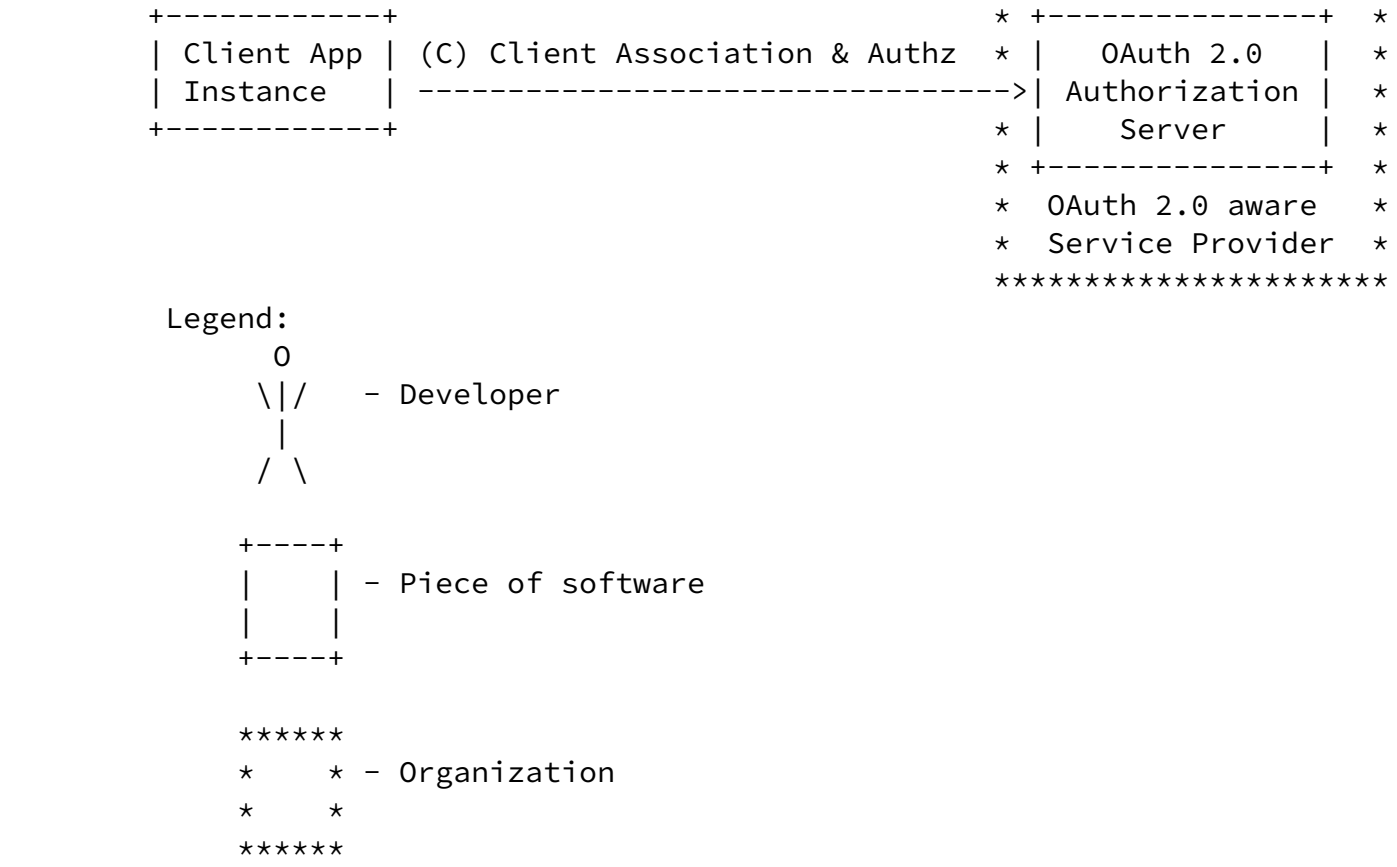


Figure 1: Client Lifecycle

(A) The client developer packages the client software with the signed software statement and distributes the client application. Local distributions may also be produced that include an initial registration token designed for use within a specific deployment domain. The method for doing this is out-of-scope of this specification.

(B) Upon receiving, or becoming aware of, a client application software distribution, an administrator configures administrative policy to accept or reject a particular client software statement within a deploying organization. Additionally an administrator may configure broader policy that accepts software by name, author, or signing organization. An administrator might also pre-approve client software by automatically accepting software statements from a particular signer or other category that can be

derived from a software statement. As part of the approval, an initial registration token may be generated for use with a local distribution of the client software (step A).

(C) To associate with a new OAuth provider, dynamic clients present a software statement, deployment specific parameters, and an optional initial registration token with a "grant_type" of "urn:ietf:params:oauth:grant-type:client-assoc". The authorization server MAY provide a client resource endpoint URL that Clients MAY use to access or update their registration. Clients wishing to rotate client credentials follow the same process except they use the client refresh token as their registration token.

Transient clients MAY perform implicit authorization requests (per section 4.2 of [RFC6749]) by submitting their software statement as the client identifier ("client_id"). Upon receiving an access token, transient client MAY then make normal resource requests.

3. Client Association

This section defines 3 types of client association and the process by which each client type associates with a software API deployment.

3.1. Static Association Clients

Clients that are written for a specific set of endpoints and do not require installation or runtime association are known as 'static clients'. These clients typically have a "client_id"(s) and client credential(s) acceptable to the deployment endpoint(s) that are integrated with the application at compilation or packaging time. The process for how this is performed is out of scope of this

specification. These clients SHOULD work using the normal OAuth2 Framework calls [RFC6749].

3.2. Dynamic Association Clients

Dynamic association defines 3 types of transactions to support the life-cycle of clients that associate on-the-fly with deployment endpoints.

- The first time a client associates, it presents its software statement, and any optional registration parameters, to the token endpoint and receives a client credential token, an optional client refresh token, and an optional client association resource endpoint URL. Clients MAY also present an initial access token in the authoirzation header as an indication of prior authorization with the authorization server.

- A client MAY update its association after a configuration change, software update, or expiration or revocation of its client credential, MAY present its client refresh token as its authorization plus the client's software statement and optional configuration parameters to receive a new client credentaial token and an optional client refresh token.

Dynamic clients association clients use the OAuth token endpoint with "grant_type" of "urn:ietf:params:oauth:grant-type:client-assoc" to submit a software statement and any per instance registration parameters. In response the registration endpoint confirms the registration by issuing a client token and an optional client refresh token. Additionally, the registration endpoint MAY provide a client resource endpoint which can be used to retrieve additional information about the client. The use and function of the client resource endpoint is out of scope of this specification.

3.2.1. Registration Request

The value of "grant_type" MUST be "urn:ietf:params:oauth:grant-type :client-assoc".

The value of the "assertion" parameter MUST contain a software statement [I-D.draft-hunt-oauth-software-statement].

The authorization header MAY be ONE OF three values:

- Omitted to indicate a new association.

- An initial access token to indicate a new assocation that has been pre-authorized.

- A client refresh token to update an existing association and to

rotate the client access token and optional client refresh token.

A non-normative, JSON encoded example of a new association request is as follows:

```
POST /token.oauth2 HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: as.example.com

{
  "grant_type"=
    "urn:ietf:params:oauth:grant-type:client-assoc"
  "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"
}
```

The update association is similar to a new association. In the update, the main difference is the client supplies the client refresh token in the "Authorization" header. In a software update (e.g. a new verson of the client software), the client MAY provide an updated or revised software statement (not shown). A non-normative, JSON encoded example of an association update request is as follows:

```
POST /token.oauth2 HTTP/1.1
Content-Type: application/json
Accept: application/json
Authorization: Bearer ey23f2.adfj230.af32-developer321
Host: as.example.com

{
  "grant_type"=
    "urn:ietf:params:oauth:grant-type:client-assoc"
  "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"
}
```

The client MAY include additional instance specific parameters defined in Section 2.2 [I-D.draft-hunt-oauth-software-statement]. If a "software_statement" is enclosed, the token server MUST ignore any attribute in the JSON request that is specified in the statement.

3.2.2. Association Processing

If an initial access token or client referesh token is presented as an authentication credential, the server MUST process the token as per section 3.2.1 of [RFC6749].

The received software statement MUST have be validated per Section 2.3 of [I-D.draft-hunt-oauth-software-statement]

If the software statement includes values for "redirect_url" and the request includes a "redirect_url" value, the request MUST be rejected. [[should this be a SHOULD?]]

Unless otherwise stated, the server SHOULD ignore any request parameter that duplicates values provided in the software statement.

For each new association, the server SHOULD generate a new "client_id" and client token. In dynamic associations, a single "software_id" will have one or more "client_id" values associated with it.

The server SHOULD NOT change the value of "client_id" if the client updates the association by presenting a client refresh token. In such a case, the "software_id" value contained in the software statement SHOULD NOT change. When an association is updated, the server MAY invalidate outstanding OAuth authorizations and access tokens issued to the client. [[OR, should the server maintain authorizations and access tokens?]]

3.2.3. Successful Association Response

After successfully processing the association request, the token server SHALL respond with the following:

client_id REQUIRED. A unique client identifier assigned to the client software instance.

token_type REQUIRED. The type of token issued by the authorization server in parameter client_token for the purpose of client authentication as described in Section 7.1 [RFC6749].

client_token REQUIRED. The client credential token issued by the authorization server. The type and usage is indicated by the parameter token_type.

expires_in RECOMMENDED. The lifetime in seconds of the access token. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated. If omitted, the authorization server SHOULD provide the expiration time via other means or document the default value.

refresh_token OPTIONAL. A client refresh token of type "bearer" which can be used to refresh a client association. The token MAY be used to update association via the token grant request and MAY be used to access the client association resource endpoint indicated in "location".

location OPTIONAL. A URI specifying the location of a resource endpoint representing the clients association with the endpoint. The type of endpoint and usage is out of scope of this specification. [[should there be a location type? e.g. SCIM Dynamic Management?]

A non-normative JSON formated response (some values clipped for readability):

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

{
 "client_id":"s6BhdRkqt3",
 "token_type":"bearer",
 "client_token":"eyJhbGciOiJFUzI1NiJ9.
```

```
            eyJpc3Mi[...omitted for brevity...].
            J9l-ZhwP[...omitted for brevity...]",
         "client_id_issued_at":2893256800,
         "expires_at":2893276800,
         "refresh_token":"mF_9.B5f-4.1JqM",
         "location":"https://scim.example.com/Clients/s6BhdRkqt3",
         "extension_parameter": "foo"
        }
```


An non-normative HoK token example (some values clipped for readability):

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

{
 "client_id":"s6BhdRkqt3",
 "token_type":"hok",
 "client_token":"eyJhbGciOiJFUzI1NiJ9.
     eyJpc3Mi[...omitted for brevity...].
     J9l-ZhwP[...omitted for brevity...]",
 "secret"="somesymetrickey",
 "client_id_issued_at":2893256800,
 "expires_at":2893276800,
 "refresh_token":"mF_9.B5f-4.1JqM",
 "location":"https://scim.example.com/Clients/s6BhdRkqt3"
}
```

3.2.4. Error Responses

When an OAuth 2.0 error condition occurs, such as the client presenting an invalid initial access token or client refresh token, the authorization server returns an error response appropriate to the OAuth 2.0 token type. This error response is defined in Section 3.2.1 of [RFC6750].

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
: The error code, a single ASCII string.

error_description
: A human-readable text description of the error for debugging.

This specification defines the following error codes:

invalid_statement The software statement presented is not a valid assertion according to [I-D.draft-hunt-oauth-software-statement].

unapproved_software The software statement presented IS NOT approved or IS NOT registered for use with the current endpoint.

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

invalid_client_metadata
: The value of one of the request parameters is invalid or is in conflict with a value in the software statement.

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 of http://sketchy.example.com
   is not allowed for this server."
}
```

3.3. Transient Association

Transient association clients access service providers for a limited relationship usually defined by the life of any access token issued. These clients are characterized by having no deployment organization issued client tokens or identifiers. Transient clients MAY use their "software_id" as the "client_id" and use their software statement as their client token according to according to section 2.2 of [I-D.ietf-oauth-jwt-bearer].

Per section 4.2 of [RFC6749], implicit flow clients SHALL provide their "software_id" as the "client_id" when making implicit authorization requests.

Per section 2.1.2 of [I-D.draft-hunt-oauth-software-statement], it is expected that the software_id SHOULD be known to the service provider as part of its software approval process and is out of scope of this specification. If the software_id is not known, the server SHOULD respond with error code "unapproved_software" per Section 3.2.4.

Transient clients SHOULD be considered as 'public clients' as defined in [RFC6749].

3.4. Client Disassociation

[[TBD - should this be supported?]]

4. IANA Considerations

The following is a parameter registration request, as defined Section 11.2, the OAuth parameters registry, of OAuth 2.0 [RFC6749]. [[NEED TO REGISTER: software_statement, redirect_uri, etc]]

5. Security Considerations

[[TO BE REVISED]]

For clients that use redirect-based grant types such as Authorization Code and Implicit, authorization servers SHOULD require clients to register their "redirect_uris"if not specified in their software statement. 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.

Clients with software statements containing "redirect_uris" MUST NOT specify a new redirect_uri during registration.

The authorization server MUST treat all client metadata, including software statements, as self-asserted. A rogue client might use the name and logo for the legitimate client, which it is trying to impersonate. 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 not been associated by the authorization server before.

Authorization servers MAY assume that registered client software sharing the same software assertion, software_id, and other metadata SHOULD have similar operational behaviour metrics. Similarly, Authorization server administrators MAY use software_id and software_version to facilitate normal change control and approval management of client software including:

- Approval of specific clients software for use with specific protected resources.

- Lifecycle management and support of specific software versions as indicated by software_version.

- Revocation of groups of client credentials and associated access tokens when support issues or security risks identified with a particular client software as identified by software_id and software_version.

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_uri", "tos_uri", "client_uri", and "policy_uri"). For instance, a rogue client could specify a registration request with a reference to a drive-by download in the "policy_uri". The

authorization server SHOULD check to see if the "logo_uri", "tos_uri", "client_uri", and "policy_uri" have the same host and scheme as the those defined in the array of "redirect_uris" and that all of these resolve to valid Web pages.

Access tokens issued to clients to facilitate update or retrieval of client registrations SHOULD be short lived.

Clients SHOULD rotate their client credentials before they expire by obtaining an access token from the authorization server using the registration scope. If a client has not successfully rotated its credential prior to expiry, the client MUST register as a new client.

If a client is deprovisioned from a server (due to expiry or de-registration), any outstanding Registration Access Token for that client MUST be invalidated at the same time. Otherwise, this can lead to an inconsistent state wherein a client could make requests to the client configuration endpoint where the authentication would succeed but the action would fail because the client is no longer valid.

Clients that are unable to retain a client credential for the life of the client instance MAY NOT register and should continue to be treated as Public clients as defined by OAuth 2.0.

6. Normative References

[I-D.draft-hunt-oauth-software-statement]
Hunt, P., Ed. and T. Nadalin, "OAuth Software Statement", .

[I-D.ietf-jose-json-web-key]
Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-14 (work in progress), July 2013.

[I-D.ietf-oauth-assertions]

Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", draft-ietf-oauth-assertions-12 (work in progress), July 2013.

[I-D.ietf-oauth-jwt-bearer]
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-06 (work in progress), July 2013.

[I-D.ietf-oauth-saml2-bearer]
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-17 (work in progress), July 2013.

[IANA.Language]
Internet Assigned Numbers Authority (IANA), "Language Subtag Registry", 2005.

[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.

[RFC5646] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

[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.

Appendix A. Acknowledgments

This draft was based upon in large part upon the work in draft-ietf-oauth-dyn-reg-14, draft-richer-oauth-dyn-reg-core-00 and draft-richer-oauth-dyn-reg-12 and WG discussions. The authors would like to thank Justin Richer and the members of the OAuth Working Group.

Appendix B. Document History

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

-00

o First draft.

Authors' Addresses

Phil Hunt (editor)
Oracle Corporation

Email: phil.hunt@yahoo.com

Tony Nadalin
Microsoft

Email: tonynad@microsoft.com