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

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

September 27, 2013

OAuth 2.0 Software Statement
draft-hunt-oauth-software-statement-00

Abstract

This specification defines a JWT authorization assertion known as a 'software statment' for use in an OAuth protected environment. A software statement is a JWT assertion used by an OAuth client to provide both informational and OAuth protocol related assertions that aid service providers to recognize OAuth client software and its expected behaviour within an OAuth Framework protected resource environment.

Status of This Memo

This Internet-Draft will expire on March 31, 2014.

Copyright Notice

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.

This specification defines a JWT authorization assertion [I-D.ietf-oauth-jwt-bearer] known as a 'software statment'. An software statement is used by an OAuth client to provide both informational and OAuth protocol [RFC6749] related assertions that aid OAuth infrastructure to both recognize client software and determine a client's expected requirements when accessing an OAuth protected resource.

Applications using software statements, may typically go through 3 phases where:

- A software statement is generated and associated with a client application.
- A service provider approves client software for use within its domain on the basic of software_id, developer, or software statment signing organization.
- And finally, where a particular instance of a client possessing a software statement associates with a particular serivce provider.

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 uses the following additional terms:

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.

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 association.

Software Statement A signed JWT authorization token [I-D.ietf-oauth-jwt-bearer] that asserts information about the client software that may be used by registration system to qualify clients for eligibility to register. To obtain a statement, a client developer may generate a client specific assertion, or a client developer may registers 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 client-to-service provider association process.

2. Software Statement

As per the introduction, a software statement is an 'authorization' bearer token (as defined in Section 2.1 of [I-D.ietf-oauth-jwt-bearer]) that carries assertions about a software that MAY be used in one or more deployment organizations and is shared by all instances of a client application.

A software statement IS NOT an authentication assertion. A software statement is a signed set of assertions fixing both OAuth related protocol values as well as informational assertions as a signed assertion from a trusted party. A deployment organization MAY use the statement to set access policy and to recognize client software during registration or association processes.

2.1. Software Statement Lifecycle

Software statements are used in 3 stages in the lifecycle of an OAuth enabled client application. A typical lifecycle flow is illustrated in Figure 1 describing when a developer obtains a software statement and how it is used within a deployment organization.

```
     Client App
     Developer
        O          (A) Obtain Software Statement       ****************
       \|/     <--------------------------------------- * Software API *
```

```
        |                                               *   Publisher  *
       / \                                              ****************
                                                               |
        |                                                      |
        |                                                      |
        |D                                                   S | A
        |i                                                   o | p
        |s                                                   f | p
        |t                                                   t | r (B)
      A |r                                                   w | o
      p |i                                                   a | v
      p |b                                                   r | a
        |u                                                   e | l
        |t                                                     |
        |i                                                     |
        |o                                                     |
        |n                                        *************|********
        v                                         *            v       *
   +------------+                                 * +---------------+  *
   | Client App | (C) Present Software Statement  * |   OAuth 2.0   |  *
   | Instance   | --------------------------------->| Authorization |  *
   +------------+                                 * |    Server     |  *
                                                  * +---------------+  *
                                                  *  OAuth 2.0 aware   *
                                                  *  Service Provider  *
                                                  **********************
    Legend:
          O
         \|/   - Developer
          |
         / \

        +----+
        |    | - Piece of software
        |    |
        +----+

        ******
        *    * - Organization
        *    *
        ******
```

Figure 1: Client Statement Lifecycle

(A) A client developer registers a clent application with a software API publisher. The software publisher, upon approval, generates a signed software statement that is returned to the developer. Alternatively, a developer may self-sign a software statement. A

self-signed statement, while weaker from a trust perspective, allows the provider to recognize and approve software (step B). A statement ensures that all registration parameters for a client are the same for every instance of a client deployed. The advantage of having the software API publisher is that deploying organizations MAY choose to pre-approve (step B) all software signed by a common trusted organization.

This protocol and procedure for issuing a software statement to the client app developer is out-of-scope of this document. This document assumes that the client app developer has obtained such a software statement already.

(B) When an administrator at a service provider obtains a software statement, the administrator configures policies to accept or reject a particular client software statement for use within a deploying organization. An administrator may also configure broader pre-approval policy that accepts software by name, author, or signing organization, or other category that can be derived from a software statement.

(C) A client instance conveys the software statement to the service provider, as described in [I-D.draft-hunt-oauth-client-association].

2.2. Statement Attributes

The following are attributes that may be included in a software statement. For each attribute defined, a qualifier (OPTIONAL, RECOMMENDED, REQUIRED) is included that indicates the usage requirement for the client. Unless otherwise stated, all client schema attributes are String based values. For example, URIs, email addresses, identifiers, are all defined as Strings.

Authorization servers MUST reject statements if it does not understand the values of any of the following singular or multi-valued attributes. An authorization server MAY override any value (including any omitted values) provided in a statement or separately during the association process and replace the requested value with a default at the server's discretion.

Extensions and profiles of this specification MAY expand this list, and authorization servers MUST accept all fields in this list. The authorization server MUST ignore any additional parameters sent by the Client that it does not understand.

2.2.1. Singular Attributes

The following is a list of attributes that MUST have only a SINGLE value in a software statement.

software_id
REQUIRED. A unique identifier that identifies the software such as a UUID. The identifier SHOULD NOT change when software version changes or when a new installation instance is detected. "software_id" is intended to help a registration endpoint recognize a client's assertion that it is a prticular piece of software. Because of this, software identifier is usually associated with a particular client name. While the OAuth"client_id"is linked to a client software deployment instance, the "software_id" is an identifier shared between all copies of the client software. Registration servers MAY use the supplied software identifier to determine whether a particular client software is approved or supported for use in the deployment domain.

software_version
RECOMMENDED. A version identifier such as a UUID or a number. Servers MAY use equality match to determine if a particular client is a particular version. "software_version" SHOULD change on any update to the client software. Registration servers MAY use the software version and identity to determine whether a particular client version is authorized for use in the deployment domain.

client_name
RECOMMENDED. A human-readable name of the client to be presented to the user. If omitted, the authorization server MAY display the raw "software_id" value to the user instead. It is RECOMMENDED that clients always send this field. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.2.4).

client_uri
RECOMMENDED. A URL of the homepage of the client software. If present, the server SHOULD display this URL to the end user in a clickable fashion. It is RECOMMENDED that clients always send this field. The value of this field MUST point to a valid Web page. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.2.4).

jwks_uri
OPTIONAL. A URL for the client's JSON Web Key Set [I-D.ietf-jose-json-web-key] document representing the client's

public keys. The value of this field MUST point to a valid JWK Set. These keys MAY also be used for higher level protocols that require signing or encryption.

logo_uri
OPTIONAL. A URL that references a logo image for the client. If present, the server SHOULD display this image to the end user during approval. The value of this field MUST point to a valid image file. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.2.4).

policy_uri
OPTIONAL. A URL that points to a human-readable policy document for the client. The authorization server SHOULD display this URL to the End-User if it is given. The Policy usually describes how an End-User's data will be used by the client. The value of this field MUST point to a valid Web page. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.2.4).

scope
OPTIONAL. A space separated list of scope values (as described in Section 3.3 [RFC6749]) that the client can use when requesting access tokens. The semantics of values in this list is service specific. If omitted, an authorization server MAY register a client with a default set of scopes.

targetEndpoint
RECOMMENDED. A generic URI of the service API the client is registering for (often set by the software API publisher). Clients requesting access to more than one target endpoint SHOULD use a separate statement for each target.

token_endpoint_auth_method
OPTIONAL. Value containing the requested authentication method for the Token Endpoint. The server MAY override the requested value. Clients MUST check for a change in value in the registration response. 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.
* "bearer": The client is will use a bearer assertion as defined in Section 4.2 of [I-D.ietf-oauth-assertions].

Additional values can be defined via the IANA OAuth Token Endpoint Authentication Methods registry Section 3.1. Absolute URIs can also be used as values for this parameter. If unspecified or omitted, the default is "bearer".

tos_uri

OPTIONAL. A URL that points to a human-readable "Terms of Service" document for the client. The authorization server SHOULD display this URL to the End-User if it is given. The Terms of Service usually describe a contractual relationship between the End-User and the client that the End-User accepts when authorizing the client. The value of this field MUST point to a valid Web page. The value of this field MAY be internationalized as described in Human Readable Client Metadata (Section 2.2.4).

2.2.2. Multi-valued Attributes

The following is a list of multi-valued attributes that may be used in a software statement.

contacts

OPTIONAL. One or more email addresses for people responsible for this client. The authorization server MAY make these addresses available to end users for support requests for the client. An authorization server MAY use these email addresses as identifiers for an administrative page for this client.

redirect_uris

RECOMMENDED. One or more redirect URI values for use in redirect-based flows such as the Authorization Code and Implicit grant types. authorization servers SHOULD require registration of valid redirect URIs for all clients that use these grant types to protect against token and credential theft attacks.

grant_types

OPTIONAL. One or more 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 type defined in OAuth JWT Bearer Token Profiles [I-D.ietf-oauth-jwt-bearer].

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

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, and 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

OPTIONAL. One or more 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, and 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.

2.2.3. 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.

2.2.4. Human Readable Client Metadata

[[This needs to be updated to be compatible with SCIM. There is a also a problem with how to limit the amount of localization data exchange for an instance registration. Note that mobile clients tend to only need one preferred language while web clients represent many clients and may have more than 20 languages to support.]]

Human-readable Client Metadata values and client Metadata values that reference human-readable values MAY be represented in multiple languages and scripts. For example, the values of fields such as "client_name", "tos_uri", "policy_uri", "logo_uri", and "client_uri" might have multiple locale-specific values in some client registrations.

To specify the languages and scripts, BCP47 [RFC5646] language tags are added to client Metadata member names, delimited by a # character. Since JSON member names are case sensitive, it is RECOMMENDED that language tag values used in Claim Names be spelled using the character case with which they are registered in the IANA Language Subtag Registry [IANA.Language]. In particular, normally language names are spelled with lowercase characters, region names are spelled with uppercase characters, and languages are spelled with mixed case characters. However, since BCP47 language tag values are case insensitive, implementations SHOULD interpret the language tag values supplied in a case insensitive manner. Per the recommendations in BCP47, language tag values used in Metadata member names should only be as specific as necessary. For instance, using "fr" might be sufficient in many contexts, rather than "fr-CA" or "fr-FR".

For example, a client could represent its name in English as ""client_name#en": "My Client"" and its name in Japanese as ""client_name#ja-Jpan-JP": "\u30AF\u30E9\u30A4\u30A2\u30F3\u30C8\u540D"" within the same registration request. The authorization server MAY display any or all of these names to the Resource Owner during the authorization step, choosing which name to display based on system configuration, user preferences or other factors.

If any human-readable field is sent without a language tag, parties using it MUST NOT make any assumptions about the language, character set, or script of the string value, and the string value MUST be used as-is wherever it is presented in a user interface. To facilitate interoperability, it is RECOMMENDED that clients and servers use a human-readable field without any language tags in addition to any language-specific fields, and it is RECOMMENDED that any human-readable fields sent without language tags contain values suitable for display on a wide variety of systems.

Implementer's Note: Many JSON libraries make it possible to reference members of a JSON object as members of an Object construct in the native programming environment of the library. However, while the "#" character is a valid character inside of a JSON object's member names, it is not a valid character for use in an object member name in many programming environments. Therefore, implementations will need to use alternative access forms for these claims. For instance, in JavaScript, if one parses the JSON as follows, "var j = JSON.parse(json);", then the member "client_name#en-us" can be accessed using the JavaScript syntax "j["client_name#en-us"]".

2.3. Software Statement Requirements

In order to create and validate a software assertion, the following requirements apply in addition to those stated in Section 3 [I-D.ietf-oauth-jwt-bearer].

1. The JWT MAY contain any claim specified in Section 2.2.
2. The JWT MUST contain an "iss" (issuer) claim that contains a unique identifier for the entity that issued and signed the JWT. The value MAY be the client developer, a software API publisher, or a third-party organization (e.g. a consortium) that would be trusted by deploying organizations.
3. The JWT MUST contain a "sub" (subject) claim that contains a unique value corresponding to the "software_id". This number is MAY be assigned by the software API publisher.
4. The JWT MUST contain an "aud" (audience) claim containing a value that is ONE of the following:

   * A value that identifies one or more software API deployments, where the client software MAY be registered.
   * A value "urn:oauth:scim:reg:generic" which indicates the assertion MAY be used with any software API deployment environment.

5. The JWT MUST contain an "exp" (expiration) claim that limits the time window during which the JWT can be used to register clients. When registering clients, the registration server MUST verify that the expiration time has not passed, subject to allowable clock skew between systems, and reject expired JWTs. The authorization server SHOULD NOT use this value to revoke an existing client registration.

3. IANA Considerations

3.1. 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.

3.1.1. 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.

3.1.2. Initial Registry Contents

The OAuth Token Endpoint Authentication Methods registry's initial contents are:

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

- Token Endpoint Authorization Method name: "bearer"
- Change controller: IETF
- Specification document(s): [[this document]]

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

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

4. Security Considerations

The authorization server MUST treat the overall software statements, as self-asserted since there is no way to prove a client is the software it asserts to be. 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.

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.

5. Normative References

[I-D.draft-hunt-oauth-client-association]
Hunt, P., Ed. and T. Nadalin, "OAuth Client Association", .

[I-D.ietf-jose-json-web-key]
Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-16 (work in progress), September 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

- First draft.

Authors' Addresses

Phil Hunt (editor)
Oracle Corporation

Email: phil.hunt@yahoo.com

Tony Nadalin
Microsoft

Email: tonynad@microsoft.com