OpenID Connect Proposal

1. Introduction

1.1 What this document is:

A detailed overview of a proposal for the OpenID Connect standard, which is intended to provide identity and session management capabilities to the OAuth2 standard.

Identity - This proposal specifies how to obtain a user identifier from an Authorization Server in a trusted manner that is suitable for federated authentication scenarios by Clients.

Session Management - This proposal specifies how to achieve trusted synchronization of signed-in state between an Authorization Server and a Client, suitable for federated authentication scenarios.

Personalization - This proposal defines a mechanism for transport of rich profile data and personalization via a standard OAuth2-protected resource called the UserInfo Endpoint. However, in its current form, the proposal does not define a schema for the UserInfo Enpoint. This is intentionally left out in this initial form of the proposal, with the understanding that a final specification would supply such a standard schema for commonly used attributes, as well as a model for extending the schema.

1.2 What it is not:

A standard draft.

1.3 Relationship with other standard drafts:

This document assumes the existence of following standards:

- OAuth2: Refers to draft 10 [1], but the intention is to refer to the final draft eventually when available.

- Signed JSON tokens: Refers to the proposal by Dirk Balfanz [2], but could easily be adapted to use with the competing proposals by Mike Jones [3] or Nat Sakimura [4]. A discussion contrasting the characteristics of the various proposals is available in [5].

- 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].

1.4 Organization:

Section 2 of this document describes the format of an OpenID Connect Session Token

Section 3 of this document describes a standard OAuth Endpoint called ‘UserInfo’.

Section 4 of this document describes how to obtain an OpenID Connect Session Token (in isolation or combination with regular access_tokens as described in the OAuth2 standard).

Section 5 describes how to perform user authentication life-cycle management using an OpenID Connect Session Token .

1.5 Not Covered In This Proposal

For now, it does not define a schema for how to obtain personalization information beyond a user identifier. The proposal calls for the definition of a basic, extensible schema for the UserInfo Endpoint.

Discovery - The proposal does not define how to perform discovery. Defining a discovery mechanism is an integral part of OpenID Connect WG charter. This proposal can easily be extended to support discovery, in at least two ways:

- A simple algorithm could be define to combine the user_id and issuer_id to arrive at a discoverable URI; or
- A schema element could be defined for the UserInfo Endpoint to convey a discoverable URL for the user.

2. OpenID Connect Session Tokens

OpenID Connect Session Tokens are Signed JSON tokens containing the following additional information in the payload portion of a Magic Signature envelope:

user_id
: REQUIRED. A String. This is a persistent label for the user that is unique for this issuer. The Client MUST NOT assume that the user_id is globally unique and MUST use the pair (issuer, user_id) to identify the user.

display_name
: OPTIONAL. A String. Suitable for greeting the user.

In addition, the Session Token inherits the following fields from Signed JSON tokens: *issuer*, *not_before*, *not_after*, and *audience*.

OpenID Connect specifies the semantics of the audience parameter as follows:

audience

REQUIRED. The client_id, as defined in Section 2 of [1]. Essentially an arbitrary identifier assigned to the Client by the issuer.

Example payload (with spaces added for readability, should be put in compact string format before Base64 encoding for encapsulation within the signed JSON Token):

```
{ "user_id": "12345", "issuer": "server.example.com", "audience": "cl
ient.example.com", "not_before": "20101231T235959Z", "not_after": "20
110101T001459Z" }
```

Editor's Note: Examples of the Signed JSON token and of the final session_token string as transmitted/received should be supplied with sample keys and algorithms.

Editor's Note: General agreement on minimal supported algorithms is not needed for interoperability, as this is already supplied by the UserInfo Endpoint. Clients are not required to parse the token format and or implement any cryptography algorithms to implement the standard.

OpenID Connect Session Tokens MAY contain additional values, as defined by OpenID Connect extensions. It is RECOMMENDED that few additional values be ever added to the tokens, as their length is limited by requirements of web applications. In particular, the length of the tokens MUST NOT make them unsuitable for transmission in HTTP headers.

OpenID Connect Session Tokens SHOULD be relatively short-lived. Their lifetime MAY be extended using the Session Management mechanism described in Section 5, allowing for loose synchronization of signed-in state between Client and Authorization Server. Clients MAY use the OpenID Connect Session Tokens to manage the signed-in status of users. They MAY deploy them as session cookies or equivalent session management mechanisms.

OpenID Connect Session Tokens are regular OAuth tokens scoped for the OpenID Connect UserInfo resource. OpenID Connect Session Tokens SHOULD NOT encode additional scopes other than the UserInfo scope.

3. The UserInfo Protected Resource (UserInfo Endpoint)

The Authorization Server MUST provide a UserInfo protected resource.

To access the UserInfo protected resource, the Client makes a standard call to the UserInfo endpoint at the Authorization Server. The request is a standard OAuth2 protected resource request, where the OpenID Connect Session Token is used as an OAuth token for the resource. See Section 5 or OAuth2 [1] for how to supply the OAuth token in a request for a protected resource. The call includes, in addition to the session_token, the following parameter:

audience
: REQUIRED. Same as in Section 2.

The Authorization Server will validate the signature and the validity interval, and will check that the supplied audience (client_id) matches the audience parameter in the token (i.e., the Authorization Server will perform steps 1-3 of Offline Validation on behalf of the client, as defined in section 4.4). Assuming the steps are successful, the Authorization Server will reply with a JSON encoded data payload, using a schema TBD. One of the required elements returned is the user_id, as defined in Section 2 above.

If the steps are not successful, an OAuth2 standard response for invalid requests should be returned. The response should indicate reason of failure by including error detail, as defined for standard OAuth2 APIs.

Example of call to UserInfo Endpoint using the header parameter method, where the client uses client_id 'client.example.com' as the audience, and the token as above:

```
GET /UserInfo?audience=client.example.com HTTP/1.1
     Host: server.example.com
     Authorization: OAuth as23Z82934s
```

Editor's Note: In the above example, we should substitute the fake token value to match the example in the previous section. Similarly in other examples throughout.

The response is a JSON Object, where the schema includes at least the following required element

user_id
: REQUIRED. Same as in Section 2.

The Authorization Server MAY return a failure mode to an online validation request if the original session of the user with the Authorization Server has terminated. However, Clients SHOULD NOT rely on online validation as a mechanism for signed-in state synchronization. Session synchronization is supported via the Session Management mechanism described in Section 5.

Editor's Note: Additional parameters MAY be returned, as defined by extensions. In particular, it is intended that this call can be used to return additional parameters, such as a display name, photo, profile URL, other identifiers, etc. The Editor suggests that the OpenIDConnect WG define standard schemas for commonly requested attributes, perhaps based on PoCo.

4. Obtaining and Validating an OpenID Connect Session Token

OpenID Connect defines how to obtain a OpenID Connect Session Token for any of the OAuth2 client profiles.

4.1 OpenID Connect End User Authorization Request

To request an OpenIDConnect token, the client adds the following parameter to a OAuth2 End-User Authorization request:

connect
: REQUIRED. The value is the string ‘authorize’.

Editor Notes:

- *Immediate mode: It is the intention that the OpenID Connect End User Authorization Request implement a mode where the Client can make the request in an invisible i-frame, where the Authorization Server will provide a failure code to indicate that user interaction to approve the request is needed. If no such mechanism is provided in an existing OAuth2 extension, it should be defined here.*

- *User Hint: It is the intention to allow the Client to specify a user_id hint. Again, this will be defined in the OpenIDConnect proposal if not elsewhere first.*

Example of an End User Authorization Request using the User Agent profile, with request elements encoded as query parameters, and the client additionally asks for a ‘calendar’ scope:

```
GET /authorize?response_type=token&client_id=client.example.com&
redirect_uri=https://client.example.com/
callback&scope=calendar&connect=authorize HTTP/1.1

Host: server.example.com
```

4.2 OpenID Connect Access Token Request

In the OAuth2 client profiles where the Access Token Request is a separate step from the End User Authorization Request, the Access Token Request step is unmodified from the OAuth2 version.

4.3 OpenID Connect Access Token Response

In addition to the parameters present in a regular OAuth2 access token response, the following is returned

session_token
: REQUIRED. An OpenID Connect Session Token, as a String.

For the example request in Section 4.1, an appropriate response would be:

```
HTTP/1.1 302 Found
Location:https://client.example.com/
callback#session_token=as23Z82934s&
access_token=298ab234xaZw2&scope=calendar
```

4.4 Offline Validation of an OpenID Connect Session Token

Client validation of an OpenID Session Token includes the steps for validating Signed JSON tokens:

1. Retrieve the issuer's key and validate the signature.
2. Base64 decode the token and verify that the current time is within the validity period defined by 'not_before' and 'not_after' parameters

In addition:

3. Validate that the 'audience' parameter matches the client_id assigned by the issuer.

4.5 Online Validation of an OpenID Session Token

The client may also validate an OpenID Session Token and retrieve the user_id by making a request to the UserInfo protected resource, using the session_token as the OAuth Token granting access to that resource, as described in Section 3.

5 OpenID Connect Session Management

The Session Management API works in a very similar manner as described in Obtaining Access Tokens in Section 4.

5.1 Session Revalidation

If the OpenID Connect Session Token is about to become invalid because the 'not_after' time is closely approaching, the Client MAY perform a session revalidation request. This is simply an authorize request as defined in session 4.1, using the immediate or non-interactive mode.

Example of a revalidation request, where 'immediate=1' indicates the fact that the request is processed without user interaction

```
GET /authorize?response_type=token&client_id=client.example.com&
redirect_uri=https://client.example.com/
callback&connect=authorize&immediate=1 HTTP/1.1

Host: server.example.com
```

The Authorization Server MUST NOT require user interaction to process the session revalidation request. The Authorization server MUST allow for the process to complete within an invisible i-frame.

Assuming the user's session with the Authorization Server is still valid, the Authorization Server

responds as described in Section 4.

```
HTTP/1.1 302 Found
Location:https://client.example.com/
callback#session_token=23zx9820234A
```

The Authorization Server MUST NOT allow the session to be revalidated if the user no longer has an authenticated session with the Server.

It is RECOMMENDED that the Client perform Session Revalidation only upon token expiration or other extraordinary circumstances, so as not to overload the Authorization Server with too-frequent requests.

The Authorization Server SHOULD issue tokens as short-lived as it is reasonable, taking in balance conflicting requirements such as the revalidation load volume it is able to handle, the latency cost for the Clients to revalidate tokens, and the desire to closely synchronize signed-in session state for superior user experience and security.

5.2 Session Invalidation

Supports Client-initiated Single sign-off.

Session invalidation requests are similar to session revalidation requests, however the connect parameter receives a different value, ‘invalidate’.

connect
: REQUIRED. The value ‘invalidate’.

Example of an invalidation request:

```
GET /authorize?response_type=token&client_id=client.example.com&
redirect_uri=https://client.example.com/signoff&connect=invalidate
HTTP/1.1
Host: server.example.com
```

The response_type parameter is supplied for compliance with OAuth2 request.

The Authorization Server MAY require that the user confirm the desire to terminate the session with the Authorization Server. The Authorization Server MAY provide an option to apply this approval automatically to subsequent session invalidation requests from the same Client.

Upon processing the invalidation request, the Authorization Server simply redirects to the callback

```
HTTP/1.1 302 Found
Location:https://client.example.com/signoff
```

Editor Note:

Immediate Mode: As in the End User Authentication Request, it is the intention to support a non-interactive mode here, where the Authorization Server MAY indicate the need for user interaction through proper error codes.

References

[1] OAuth2, draft 10: http://tools.ietf.org/html/draft-ietf-oauth-v2-10
[2] Signed JSON tokens, by D. Balfanz: http://balfanz.github.com/jsontoken-spec/draft-balfanz-jsontoken-00.html
[3] Signed JSON tokens, by M. Jones: http://self-issued.info/docs/draft-goland-json-web-token-00.html
[4] Signed/Encrypted JSON tokens, by N. Sakimura: http://jsonenc.info/jss/1.0/
[5] Comparing the JSON token drafts: http://www.ietf.org/mail-archive/web/oauth/current/msg04590.html