OpenID Connect 1.0 for Enterprise

PAUL MADSSEN
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>EXECUTIVE OVERVIEW</td>
</tr>
<tr>
<td>04</td>
<td>WHAT IS OPENID CONNECT?</td>
</tr>
<tr>
<td></td>
<td>Connect Terminology</td>
</tr>
<tr>
<td></td>
<td>Relationship to OAuth</td>
</tr>
<tr>
<td></td>
<td>Relationship to SAML</td>
</tr>
<tr>
<td></td>
<td>Trust Model</td>
</tr>
<tr>
<td></td>
<td>Discovery</td>
</tr>
<tr>
<td></td>
<td>Dynamic Registration</td>
</tr>
<tr>
<td></td>
<td>Flows</td>
</tr>
<tr>
<td>10</td>
<td>USE CASES</td>
</tr>
<tr>
<td></td>
<td>Outbound Web SSO</td>
</tr>
<tr>
<td></td>
<td>Inbound Web SSO</td>
</tr>
<tr>
<td></td>
<td>Employee Accesses On-premises API via Native Mobile Application</td>
</tr>
<tr>
<td></td>
<td>Employee Accesses SaaS API via Native Mobile Application</td>
</tr>
<tr>
<td></td>
<td>Use with Provisioning Protocols (SCIM)</td>
</tr>
<tr>
<td>13</td>
<td>CONCLUSION</td>
</tr>
</tbody>
</table>
EXECUTIVE OVERVIEW

In order to meet the challenges presented by the use of mobile apps and cloud services in the enterprise, a new generation of identity protocols has been developed.

OpenID Connect 1.0 uses the same building blocks as the modern application architectures and APIs it secures—REST and JSON. This allows it to profile and extend OAuth 2.0 to add an identity layer—creating a singular, cohesive framework that promises to secure APIs, mobile native applications and browser applications.

This paper introduces OpenID Connect (Connect), detailing its features and its application in enterprise use cases.
OpenID Connect is a simple JSON-/REST-based identity protocol built on top of the OAuth 2.0 and JWT family of protocols. Its design philosophy is to "keep simple things simple and make complicated things possible."

The goal was to develop a single protocol that provides:

- Mobile-friendly support of native applications and generally restricted bandwidth and capabilities of feature phones.
- A single, seamless flow for integrating API authorization and user authentication.
- Lightweight implement for applications and relying party (RP) websites, replacing proprietary last mile integrations.

Connect reflects years of experience with the strengths and weaknesses of earlier and existing protocols. It’s strongly influenced by the enterprise experiences of SAML, ID-WSF and the WS* stack (including Information Cards), but it uses more modern components like REST and JSON.

Connect also reflects community experiences in dynamic configuration and connection scalability, learned from the consumer-centric specifications of OpenID 1.0 and 2.0 specifications. For example, whereas SAML deployments are typically based on static and pre-established trust, OpenID allowed for a trust relationship to be dynamically established between identity providers and RPs.

The following diagram represents what has influenced Connect, showing that its development has been driven by both consumer-centric and enterprise-centric influences. As a result, it promises to address the requirements of both deployment models.

Motivated by the adoption of Connect by RPs, part of Connect’s design philosophy was to push complexity to its identity provider wherever possible. Compared to SAML, the burden of supporting the protocol is much more evenly shared between the identity provider and the service provider. Because of this asymmetry in support complexity, Connect applies much more easily to native mobile applications, which are far more constrained than server implementations.
One of the key areas of simplification from SAML was the migration from XML to JSON. Because it’s the data representation format used by JavaScript in all modern web browsers, JSON has wide support in all modern programming environments.

The JSON Web Token (JWT) format, including its methods for signing and encryption (JOSE), allows for more compact security tokens than are possible with XML (and realized by SAML). Small security tokens are required to fit the URL and header size constraints in mobile environments. The JWT format also avoids the problems that XML signing and encryption have been plagued with over their history.

The Connect protocol suite is modular and only a subset needs to be implemented.

Basic client and implicit client profiles are quick start guides for those who want to build Connect client capabilities into their applications.

The specifications are published on the OpenID foundation website.

Connect was developed under a mutual non-assert IPR agreement by all of the participants in the specification, including Ping Identity®, Google Inc., Microsoft®, PayPal®, Facebook®, IBM®, CA® and many others. This allows implementations to be built royalty-free by developers.
CONNECT TERMINOLOGY

Because it’s built on top of OAuth 2, Connect uses much of the same terminology and definitions.

- **Subject**: The user that is requesting access to a protected resource, either through a web interface or via API calls.
- **Resource Server (RS)**: The server hosting the protected resources.
- **Authorization Server (AS)**: The server issuing access and identity tokens after successfully authenticating the subject.
- **Client**: An application requesting tokens from the AS (identity tokens to enable subject access to client-hosted resources, or access tokens to enable access to a RS-hosted resource).
- **Discovery Endpoint**: Makes configuration information available that describes the Connect AS and how to interact with it.
- **Registration Endpoint**: Allows new clients to dynamically register with the Connect AS.
- **Access Token**: Identity and authorization assertion issued by the AS and consumed by the RS.
- **ID Token**: Identity and authentication assertion issued by the AS and consumed by the client.

RELATIONSHIP TO OAUTH

OAuth 2.0 is a framework for API authentication and authorization. It defines how an API client can obtain security tokens that express a set of permissions against that API. These tokens are attached by the client to its API, and they indicate the client’s authorizations.

OpenID Connect provides identity semantics and constructs on top of OAuth, logically layering identity onto the OAuth base. Connect specifies how to use OAuth to ‘do’ identity, as opposed to other non-identity centric applications that are possible with OAuth.

Connect closes some of the interoperability and scaling gaps of OAuth, and it introduces two notable identity constructs on top of the OAuth base protocol:

1. **The id_token**: a structured JWT that allows the AS to assert the authentication status of a subject to the client (and possibly other identity attributes). The delivery of the id_token from the AS to the client enables web single sign-on (SSO) for the subject when they visit the client. The id_token may optionally be encrypted for confidentiality, and it’s constructed from a base64url encoded signature description (the algorithm), a period (.), a base64url encoded JSON object, a period (.), and a signature string. The id_token typically includes:
   - The unique subject (user) identifier.
   - The issuing authority (the AS URI).
   - The intended audience (the client application).
   - Issue and expiration times.
   - OPT how the user was authenticated.

2. **The UserInfo endpoint**: a standardized REST API for identity attributes. By calling this OAuth-protected API, the client can obtain the identity attributes it would otherwise collect in a registration form, such as email, name, address, etc. The UserInfo endpoint is a regular OAuth 2.0 resource that returns a JSON document when fetched via HTTP. The client constructs an HTTPS “GET” request to the UserInfo endpoint and includes the access token in the authorization header.
The typical OAuth deployment model has one AS issuing tokens for some number of protected RS that are in the same administrative domain. Because of this assumption, OAuth initially defines no standardized mechanisms by which an RS can validate access tokens on the API calls it receives. This was effectively left as a deployment exercise. But in practice, it means that deployments and commercial implementations of OAuth create their own proprietary introspection mechanisms that the RS can call back to the local AS to validate access tokens. Connect’s definition of how to use structured JWT for identity tokens helped spur the OAuth group to standardize structured access tokens that don't require introspection; the RS can validate access tokens locally through cryptographic checks.

Additionally, clients in OAuth are typically registered at the AS through a manual administrative process via an AS-hosted web page. Through this process, (a) the AS issues credentials to the client for use on the OAuth messages to follow, and (b) the client obtains the relevant AS metadata (like endpoint URLs) necessary for it to interact with that AS. Connect allows for both of these steps to be conducted in a more automated and dynamic fashion—dynamic registration allows clients to register themselves at the AS without manual intervention, and a discovery mechanism allows clients to dynamically obtain the necessary AS configuration information.

RELATIONSHIP TO SAML

SAML and Connect have similar SSO flows, supporting both the redirection of assertions through the browser and through a back channel via an artifact dereferencing step. Both use a signed assertion to convey claims about a subject and session information such as the authentication method and context.

Connect also has many features that allow both easy connection between partners and the acceptance of external social identities via connections that are dynamically provisioned on demand.

Connect also provides the client/SP with API access tokens in the same flow, allowing it to support dynamic provisioning of account information and to make ongoing API calls on behalf of the user. This is similar to, but more REST friendly, than the SAML attribute query.

Connect also includes a cross-organization trust model similar to SAML, allowing it to be used in place of SAML between organizations for federation. Existing SAML-based federation may transition to Connect over time.
TRUST MODEL

The Connect trust model between the AS and the client is similar to that of SAML. The subject’s authentication status is expressed in the JWT-based id_token, which is signed by the AS. The default signature mechanism is an RSA signature (as in SAML), though other types of signatures may be used for performance and size if necessary. Upon delivery of the id_token to the client, the client validates the signature before making an authorization decision regarding the subject’s access request.

The main difference between the SAML and Connect trust models is the use of dynamically discovered and provisioned metadata.

The trust model between the RS and AS is similar to that of OAuth, although Connect’s support for JWT-formatted access tokens allows for an RS to validate tokens locally, as opposed to an introspection step.

<table>
<thead>
<tr>
<th>SAML</th>
<th>OAuth</th>
<th>Connect</th>
</tr>
</thead>
<tbody>
<tr>
<td>IdP/AS and SP/Client</td>
<td>Signed XML assertions</td>
<td>Signed JWT tokens</td>
</tr>
<tr>
<td>AS and RS</td>
<td>Token Introspection</td>
<td>Token Introspection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Signed JWT tokens</td>
</tr>
</tbody>
</table>

DISCOVERY

For all web SSO protocols, a fundamental challenge for the RP is to determine which is the appropriate identity provider for a given user. In Connect, the client must work out the relevant AS for a given subject. Generally, the client depends on the subject to facilitate this step. A variety of approaches exist, from the NASCAR model of multiple icons for the user to pick from (implying that the client may have already worked out the technical details with the AS) or, more flexibly, asking the subject to provide to the client their email address or some other identifier.

This happens in the SaaS world when an enterprise employee presents themselves to the SaaS login page (as opposed to following a link). The challenge for the SaaS is three-fold. (1) Is this employee from an enterprise customer that leverages SSO (as opposed to passwords)? (2) What SSO protocol does the enterprise use (e.g., SAML or Connect)? (3) If they use Connect, what’s the address of the enterprise AS authorization endpoint to direct the browser to?

Connect discovery defines how to leverage the Webfinger specification to turn a user-provided identifier into the address of the corresponding AS configuration file that includes the addresses for the different Connect endpoints with which the client will interact.
DYNAMIC REGISTRATION

Dynamic registration allows the client to register itself with the AS and receive a client identifier and secret—the credentials used when making subsequent authorization and token requests to the AS. The registration endpoint can be restricted to authorized clients via an access token.

Typically, to enable a Connect link to a SaaS in support of enterprise employees being able to web SSO into the SaaS, the enterprise would provide the SaaS with its AS URI and a registration token as part of an administrative registration process provided by the SaaS. The SaaS would then perform discovery on the issuer URI to find the registration endpoint. Using the registration token provided by the enterprise, the SaaS would push its configuration to the registration endpoint and receive a confirmation of it’s client identifier, secret and configuration details.

As Connect clients can register directly with the AS, Connect allows for automated negotiation of features and parameters on a per-AS basis.

FLOWs

Below is a typical Connect flow. The AS issues two different tokens to the client. The id_token enables SSO for the subject into the client and the access token allows the client to make API calls to both the Connect-defined UserInfo API as well as other application APIs.

The identity features that Connect introduces to the normal OAuth 2 flow are highlighted above.

1. The transfer of the id_token from the AS to the client enables web SSO.
2. The UserInfo endpoint is a standardized REST API from which the client can obtain profile data for the user.
A more complicated scenario would have the client use the registration and discovery mechanisms before interacting with the AS authorization endpoint, as shown below. From this point on, the flow would be the same as shown in the previous diagram.

**USE CASES**

**OUTBOUND WEB SSO**

Partners or SaaS that have traditionally allowed enterprise SSO via SAML are now allowing enterprise users to log in to SSO-enabled SaaS using Connect as well. In this use case, enterprise users access Connect-enabled SaaS web applications and enable SaaS access to enterprise APIs with a single login.

1. User logs in to the enterprise using corporate credentials.
2. Enterprise AS delivers an id_token and access token to the SaaS client.
3. SaaS provides application access to the subject.
4. OPT SaaS client uses access token on API calls to enterprise.
This model:

- Provides employees access to partner or SaaS web applications.
- Avoids exposing enterprise passwords.
- Simplifies employee enrollment.
- Optionally provides the SaaS with secure access to enterprise APIs.

**INBOUND WEB SSO**

Users with social identities can access consumer-facing, web-based applications (provided by banks, ISPs, retailers, etc.) via a browser using Connect. Such enterprises might choose to allow this scenario to minimize the burden associated with initial interactions with a potential customer with the goal of driving customer acquisition. By leveraging the user’s existing social identities, the enterprise makes it easier for that potential customer to browse personalized offerings. If and when the user becomes a (paying) customer, the enterprise may choose to issue their own identity and credentials, and either cancel the connection to the social provider or maintain it for the attribute-sharing component (if not, for SSO alone).

1. User logs in to their preferred social provider using existing credentials.
2. Social provider AS delivers an id_token and access token to the enterprise client.
3. Enterprise provides application access to the subject.
4. Enterprise client uses an access token on API calls to the social provider requesting data or, for instance, posting a tweet on behalf of the user.

This model:

- Enables third-party access to web applications.
- Avoids storing passwords for external users.
- Avoids costs of managing user passwords.
- Allows users to select their preferred social provider for different contexts.

Of course, the model isn’t limited to third-party social providers. It also applies to more general third-party provided identity, such as from business partners and customers.
EMPLOYEE ACCESSES ON-PREMISE API VIA NATIVE MOBILE APPLICATION

1. User logs in to the enterprise using corporate credentials.
2. Enterprise AS delivers an access token to the native application client.
3. Native application client uses the access token on API calls to the enterprise.
4. Enterprise API returns the application data to the native application.

EMPLOYEE ACCESSES SAAS API VIA NATIVE MOBILE APPLICATIONS

Enterprise employees are more and more often accessing SaaS functionality via APIs accessed from a native mobile application. In this scenario, two different Connect flows are sequenced. The first is a web SSO flow from the enterprise to a SaaS. The second is an API authorization flow from the same SaaS to its own native application installed on an employee's device.

1. User logs in to the enterprise using their corporate credentials.
2. Enterprise AS delivers an id_token to the SaaS client.
3. SaaS AS creates and delivers an access token to its native application client.
4. SaaS native application client uses an access token on API calls to the SaaS.
5. SaaS API returns the application data to the native application.
The first Connect flow (step #2 above) could also be a SAML flow. They are effectively interchangeable in this context, and importantly, independent of the OAuth flow that follows.

If distributed through the public application stores, a SaaS native application will be unlikely to have a unique client secret to use to authenticate to the AS when obtaining tokens (step #3 above). Consequently, there is a risk of a malicious native application inserting itself into the issuance flow and gaining possession of access tokens and the permissions associated with them. The Proof Key Code Exchange (PKCE) specification mitigates this risk by effectively allowing the native application to dynamically generate transient client secrets that can be used to authenticate to the AS.

USE WITH PROVISIONING PROTOCOLS (SCIM)

The System for Cross-Domain Identity Management (SCIM) 2.0 defines a REST protocol for user attribute management. When used to read user attributes, SCIM is comparable to Connect’s UserInfo API, although using a different JSON schema. SCIM also allows for its API calls to be protected through OAuth access tokens, so it would be possible to use the Connect authorization flow to deliver an access token to the client. This access token is then to be used on a SCIM API call.

CONCLUSION

OpenID Connect logically combines the functionality of SAML and OAuth into a single, cohesive and comprehensive framework. While SAML has enjoyed significant success as an SSO protocol in the enterprise, SaaS and higher education, it has limited support for dynamic trust and configuration. It’s also seen no adoption in the consumer space, and its attribute-sharing mechanisms haven’t been widely deployed. Similarly, while OAuth has emerged as a powerful authorization mechanism for APIs in both the consumer and enterprise spaces, it too presumes a static trust and configuration model and has no explicit concept of identity. Connect addresses these limitations of SAML and OAuth—and does so with a modern REST and JSON-based architecture.