OpenID Connect explained

<http://connect2id.com/learn/openid-connect>

*OpenID Connect is the new emerging standard for single sign-on and identity provision on the internet. Its formula for success: simple JSON-based identity tokens (JWT), delivered via the OAuth 2.0 protocol to suit web, browser-based and native / mobile apps.*

[**1. Local user authentication vs Identity Providers**](http://connect2id.com/learn/openid-connect#intro)

Applications often need to identify their users. The simplistic approach is to create a local database for the users’ accounts and credentials. Given enough technical care this can be made to work well. However, local authentication can be bad for business:

* People find sign up and account creation tedious, and rightly so. Consumer web sites and apps may suffer abandoned shopping carts because of that, which means loss of business and sales.
* For enterprises with many apps, maintenance of separate user databases can easily become an administrative and security nightmare. You may want to put your IT resources to better use.

The established solution to these problems is to delegate user authentication and provisioning to a dedicated, purpose-built service, called an Identity Provider (IdP). Google, Facebook and Twitter, where many people on the internet are registered, offer such IdP services for their users. A consumer web site can greatly streamline user onboarding by integrating login with these IdPs.

In an enterprise, this would ideally be one internal IdP service, for employees and contractors to log into internal apps. Centralization has considerable benefits, such as easier administration and potentially faster development cycles for new apps. You may ask: Isn’t that going to create a single point of failure? No, not when the IdP service is built for redundancy.

[**2. Enter OpenID Connect**](http://connect2id.com/learn/openid-connect#oidc)

[OpenID Connect](http://openid.net/specs/openid-connect-core-1_0.html), published in 2014, is not the first standard for IdP, but definitely the best in terms of usability and simplicity, having learned from lessons from past efforts such as [SAML](https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language) and OpenID 1.0 and 2.0.

What is the formula for success of OpenID Connect?

* **Easy to consume identity tokens**: Client apps receive the user’s identity encoded in a secure JSON Web Token (JWT), called ID token. JWTs are appreciated for their elegance and portability, with support for a range of signature and encryption algorithms. All this makes JWT outstanding for the job of ID tokens.
* **Based on the OAuth 2.0 protocol**: Clients use OAuth 2.0 flows to obtain ID tokens, designed to fit web apps as well as native / mobile apps. OAuth 2.0 also means one protocol for authentication and authorisation (obtaining access tokens).
* **Simplicity**: OpenID Connect is simple enough to integrate with basic apps, but it also offers a number of features and security options to match demanding enterprise requirements.

[**3. The identity token**](http://connect2id.com/learn/openid-connect#id-token)

The [ID token](http://openid.net/specs/openid-connect-core-1_0.html#IDToken) resembles the concept of an identity card, in a standard [JWT](http://tools.ietf.org/html/rfc7523) format, signed by the OpenID Provider (OP). To obtain one the client needs to send the user to their OP with an authentication request.

Features of the ID token:

* Asserts the identity of the user, called *subject* in OpenID (sub).
* Specifies the issuing authority (iss).
* Is generated for a particular *audience*, i.e. client (aud).
* May contain a nonce (nonce).
* May specify when (auth\_time) and how, in terms of strength (acr), the user was authenticated.
* Has an issue (iat) and an expiration date (exp).
* May include additional requested details about the subject, such as name and email address.
* Is digitally signed, so it can be verified by the intended recipients.
* May optionally be encrypted for confidentiality.

The ID token statements, or claims, are packaged in a simple JSON object:

{

"sub" : "alice",

"iss" : "https://openid.c2id.com",

"aud" : "client-12345",

"nonce" : "n-0S6\_WzA2Mj",

"auth\_time" : 1311280969,

"acr" : "c2id.loa.hisec",

"iat" : 1311280970,

"exp" : 1311281970,

}

The ID token header, claims JSON and signature are encoded into a base 64 URL-safe string, for easy passing around, for example as URL parameter.

eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzcyI6ICJodHRw

Oi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5NzYxMDAxIiw

KICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZfV3pBMk1qIi

wKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5NzAKfQ.ggW8hZ

1EuVLuxNuuIJKX\_V8a\_OMXzR0EHR9R6jgdqrOOF4daGU96Sr\_P6qJp6IcmD3HP9

9Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJNqeGpe-gccM

g4vfKjkM8FcGvnzZUN4\_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7TpdQyHE5lcMiKP

XfEIQILVq0pc\_E2DzL7emopWoaoZTF\_m0\_N0YzFC6g6EJbOEoRoSK5hoDalrcvR

YLSrQAZZKflyuVCyixEoV9GfNQC3\_osjzw2PAithfubEEBLuVVk4XUVrWOLrLl0

nx7RkKU8NXNHq-rvKMzqg

You can read more about the JWT data structure and encoding in [RFC 7519](http://tools.ietf.org/html/rfc7519).

[**4. How to request an ID token**](http://connect2id.com/learn/openid-connect#auth-request)

Now that we know what an [ID token](http://connect2id.com/learn/openid-connect#id-token) is, how can a client, called Relying Party (RP) in OpenID Connect, request one?

Authentication must take place at the identity provider, where the user’s session or credentials will be checked. For that a trusted agent is required, and this role is usually played by the web browser.

Browser popups is the preferred way for web apps to redirect the user to the IdP. Mobile apps on platforms such as Android or iOS should launch the system browser for that purpose. Embedded web views are not be trusted, as there’s nothing to prevent the app from snooping on the user password. User authentication must always occur in a trusted context separate from the app’s context (e.g. the browser).

Note that OpenID Connect doesn’t specify how users should actually be authenticated, this is [left up to providers](http://connect2id.com/learn/openid-connect#auth-methods).

ID tokens are requested via the OAuth 2.0 protocol ([RFC 6749](http://tools.ietf.org/html/rfc6749)), which has been a tremendous success on its own. OAuth was originally devised as a simple authorization mechanism for apps to obtain access tokens for web APIs or other protected resources. It has *flows* designed for all app types: traditional server-based web apps, browser (JavaScript) only apps, and native / mobile apps.

So what are those flows, or paths, for obtaining ID tokens?

* [Authorisation code flow](http://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth) — the most commonly used flow, intended for traditional web apps as well as native / mobile apps. Involves an initial browser redirection to / from the OP for user authentication and consent, then a second back-channel request to retrieve the ID token. This flow offers optimal security, as tokens are not revealed to the browser and the client app can also be authenticated.
* [Implicit flow](http://openid.net/specs/openid-connect-core-1_0.html#ImplicitFlowAuth) — for browser (JavaScript) based apps that don’t have a backend. The ID token is received directly with the redirection response from the OP. No back-channel request is required here.
* [Hybrid flow](http://openid.net/specs/openid-connect-core-1_0.html#HybridFlowAuth) — rarely used, allows the app front-end and back-end to receive tokens separately from one another. Essentially a combination of the code and implicit flows.

The OpenID Connect spec provides a nice [comparison](http://openid.net/specs/openid-connect-core-1_0.html#Authentication) of the three flows, reproduced here in a simplified form.

|  |  |  |  |
| --- | --- | --- | --- |
| **Flow property** | **Code** | **Implicit** | **Hybrid** |
| Browser redirection step | ✔ | ✔ | ✔ |
| Backend request step | ✔ | ✕ | ✔ |
| Tokens revealed to browser | ✕ | ✔ | ✔ |
| Client can be authenticated | ✔ | ✕ | ✔ |

[**5. Cool ID token uses**](http://connect2id.com/learn/openid-connect#cool-id-token-uses)

The [ID token](http://connect2id.com/learn/openid-connect#id-token) is versatile, and its use is certainly not limited to just signing in users into apps:

* **Stateless sessions** — Put into a browser cookie the ID token can be used to implement lightweight stateless sessions. This does away with the need to store sessions on the server side (in memory or on disk), which can be quite a burden for apps that must scale well. The session cookie is checked by validating the ID token. If the token has expired the app can simply ask the OP for a new one via a silent *prompt=none* request.
* **Passing identity to 3rd parties** — The ID token may be passed to other components of the app or to backend services when knowledge of the user’s identity is required, for example to log audit trails.
* **Token exchange** — The ID token may be exchanged for an access token at the token endpoint of an OAuth 2.0 authorisation server ([draft-ietf-oauth-token-exchange-03](https://tools.ietf.org/html/draft-ietf-oauth-token-exchange-03)). There are many real world scenarios when an identity document is required to obtain access, for example when you check in at a hotel to get your room key. Token exchange has uses in distributed and enterprise applications.

[**6. Example OpenID authentication**](http://connect2id.com/learn/openid-connect#example-auth-code-flow)

We will now go through a minimal example of how to obtain an ID token for a user from an OP, using the [authorization code flow](http://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth). This is the most commonly used flow by traditional web apps and native mobile apps. Examples of the [implicit](http://openid.net/specs/openid-connect-core-1_0.html#ImplicitFlowAuth) and [hybrid flow](http://openid.net/specs/openid-connect-core-1_0.html#HybridFlowAuth) can be found in the [OpenID Connect spec](http://openid.net/specs/openid-connect-core-1_0.html).

The code flow has two steps:

|  |  |  |
| --- | --- | --- |
|  | **Step 1** | **Step 2** |
| **Purpose** | 1. Authenticate user 2. Receive user consent | 1. Authenticate client (optional) 2. Exchange code for token(s) |
| **Via** | Front-channel request (browser redirection) | Back-channel request (app to web server) |
| **To** | Authorisation endpoint | Token endpoint |
| **Result on success** | Authorisation code (step 2 input) | ID token (+ OAuth 2.0 access token) |

[***Code flow: Step 1***](http://connect2id.com/learn/openid-connect#example-auth-code-flow-step-1)

The RP initiates user authentication by redirecting the browser to the [OAuth 2.0 authorization endpoint](http://tools.ietf.org/html/rfc6749" \l "section-3.1) of the OpenID Provider. The OpenID [authentication request](http://openid.net/specs/openid-connect-core-1_0.html#AuthRequest) is essentially an OAuth 2.0 [authorization request](http://tools.ietf.org/html/rfc6749#section-4.1.1) to access the user’s identity, indicated by an openid value in the scope parameter.

Example authentication redirection to the OP:

HTTP/1.1 302 Found

Location: https://openid.c2id.com/login?

response\_type=code

&scope=openid

&client\_id=s6BhdRkqt3

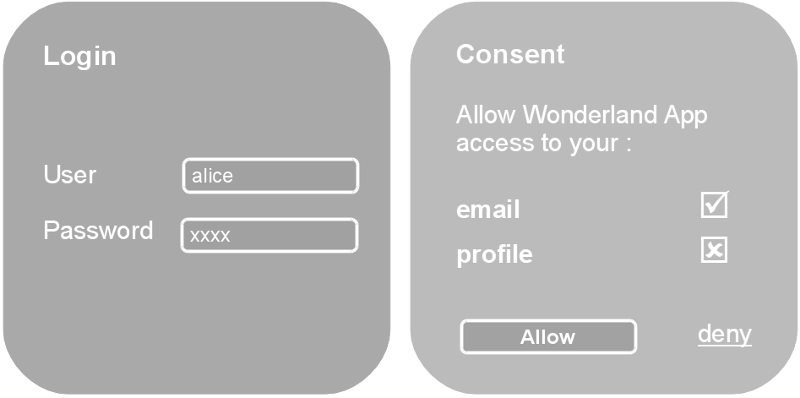
&state=af0ifjsldkj

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

The request parameters are encoded in the URI query:

* **response\_type** Set to code to indicate an authorization code flow.
* **scope** Used to specify the scope of the requested authorization in OAuth. The scope value openid signals a request for OpenID authentication and ID token.
* **client\_id** The client identifier of the RP at the OP. This identifier is typically obtained when the RP is registered with the OP, via the client registration API, developer console, or some other method.
* **state** Opaque value set by the RP to maintain state between request and callback.
* **redirect\_uri** The RP callback URI for the authentication response.

At the OP, the user will typically be authenticated by checking if they have a valid session (established by a browser cookie), and in the absence of that, by prompting the user to login. After that the user will typically be asked whether they agree to sign into the RP.



The OP will then call the client redirect\_uri with an authorization code (on success) or an error code (if access was denied, or some other error occurred, such a malformed request was detected).

HTTP/1.1 302 Found

Location: https://client.example.org/cb?

code=SplxlOBeZQQYbYS6WxSbIA

&state=af0ifjsldkj

The RP must validate the state parameter, and use the code to proceed into the [next step](http://connect2id.com/learn/openid-connect#example-auth-code-flow-step-2) - exchanging the code for the ID token.

[***Code flow: Step 2***](http://connect2id.com/learn/openid-connect#example-auth-code-flow-step-2)

The authorization code is an intermediate credential, which encodes the authorization obtained at [step 1](http://connect2id.com/learn/openid-connect#example-auth-code-flow-step-1). It is therefore opaque to the RP and only has meaning to the OP server. To retrieve the ID token the RP must submit the code to the OP, but this time a direct back-channel request is needed.

All this is done for two reasons:

* To authenticate confidential clients with the OP before revealing the tokens;
* To deliver the tokens straight to the RP, thus avoid exposing them to the browser.

The code-for-token exchange happens at the [token endpoint](http://connect2id.com/learn/openid-connect#token-endpoint) of the OP:

**POST** /token HTTP/1.1

Host: openid.c2id.com

Content-Type: application/x-www-form-urlencoded

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant\_type=authorization\_code

&code=SplxlOBeZQQYbYS6WxSbIA

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

The client ID and secret are passed via the Authorization header. Apart from [HTTP basic authentication](http://tools.ietf.org/html/rfc6749#section-2.3.1) OpenID Connect also permits authentication with signed [JWT assertions](http://tools.ietf.org/html/rfc7523), which do not expose the client secret with the token request, and hence offer better security.

The token request parameters are form-encoded:

* **grant\_type** Set to authorization\_code.
* **code** The code obtained from [step 1](http://connect2id.com/learn/openid-connect#example-auth-code-flow-step-1).
* **redirect\_uri** Repeats the callback URI from step 1.

On success the OP will return a JSON object with the ID token, an access token and an optional refresh token:

HTTP/1.1 200 OK

Content-Type: application/json

Cache-Control: no-store

Pragma: no-cache

{

"id\_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc

yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5

NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ

fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz

AKfQ.ggW8hZ1EuVLuxNuuIJKX\_V8a\_OMXzR0EHR9R6jgdqrOOF4daGU96Sr\_P6q

Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ

NqeGpe-gccMg4vfKjkM8FcGvnzZUN4\_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd

QyHE5lcMiKPXfEIQILVq0pc\_E2DzL7emopWoaoZTF\_m0\_N0YzFC6g6EJbOEoRoS

K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3\_osjzw2PAithfubEEBLuVVk4

XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"

"access\_token": "SlAV32hkKG",

"token\_type": "Bearer",

"expires\_in": 3600,

}

The ID token is a JWT and must be carefully [validated](http://openid.net/specs/openid-connect-core-1_0.html#IDTokenValidation) by the RP before it can be accepted.

Note that a [bearer access token](http://tools.ietf.org/html/rfc6750) is also included. This is to ensure the response is compliant with the [OAuth 2.0](http://tools.ietf.org/html/rfc6749) spec. For basic OpenID authentication requests where only an ID token is requested this access token is nominal and may be safely ignored. The access token however comes into play when also requesting access to user profile data at the UserInfo endpoint. More on this in the [next chapter](http://connect2id.com/learn/openid-connect#claims).

[**7. Claims (user info)**](http://connect2id.com/learn/openid-connect#claims)

OpenID Connect specifies a set of [standard claims](http://openid.net/specs/openid-connect-core-1_0.html#StandardClaims), or user attributes. They are intended to supply the client app with consented user details such as email, name and picture, upon request. [Language tags](http://en.wikipedia.org/wiki/IETF_language_tag) enable localization.

| **Scope value** | **Associated claims** |
| --- | --- |
| email | email, email\_verified |
| phone | phone\_number, phone\_number\_verified |
| profile | name, family\_name, given\_name, middle\_name, nickname, preferred\_username, profile, picture, website, gender, birthdate, zoneinfo, locale, updated\_at |
| address | address |

Clients can request claims in two ways:

* An entire claims category by its scope value (see the above table for the scope value to claim mappings)
* Individually, with the optional [claims](http://openid.net/specs/openid-connect-core-1_0.html#ClaimsParameter) request parameter.

Example request to access the user’s identity (ID token) and email, by simply setting the scope to openid email:

HTTP/1.1 302 Found

Location: https://openid.c2id.com/login?

response\_type=code

&scope=openid%20email

&client\_id=s6BhdRkqt3

&state=af0ifjsldkj

&redirect\_uri=https%3A%2F%2Fclient.example.org%2Fcb

If you’re an app developer, respect the user’s privacy and keep the requested claims down to the very essential. This will increase your chances of user conversion and will also ensure compliance with recent privacy laws.

Note that even if a user allows an app to access their identity, they may choose to deny release of some claims. The app should be able to handle such decisions gracefully.

The claims format is JSON:

{

"sub" : "alice",

"email" : "alice@wonderland.net",

"email\_verified" : true,

"name" : "Alice Adams",

"given\_name" : "Alice",

"family\_name" : "Adams",

"phone\_number" : "+359 (99) 100200305",

"profile" : "https://c2id.com/users/alice",

"https://c2id.com/groups" : [ "audit", "admin" ]

}

OpenID providers may extend the standard JSON claims schema to include [additional attributes](http://openid.net/specs/openid-connect-core-1_0.html#AdditionalClaims). Enterprises may for instance define claims such as employee role, manager, and department. The names of any additional claims should be prefixed by a URL to create a safe namespace for them and prevent collisions.

[**8. OpenID Connect provider endpoints**](http://connect2id.com/learn/openid-connect#endpoints)

This chapter summarizes the endpoints of an OpenID Connect server.

|  |  |
| --- | --- |
| **Core endpoints** | **Optional endpoints** |
| * [Authorization](http://connect2id.com/learn/openid-connect#authz-endpoint) * [Token](http://connect2id.com/learn/openid-connect#token-endpoint) * [UserInfo](http://connect2id.com/learn/openid-connect#userinfo-endpoint) | * [WebFinger](http://connect2id.com/learn/openid-connect#optional-endpoints) * [Provider metadata](http://connect2id.com/learn/openid-connect#optional-endpoints) * [Provider JWK set](http://connect2id.com/learn/openid-connect#optional-endpoints) * [Client registration](http://connect2id.com/learn/openid-connect#optional-endpoints) * [Session management](http://connect2id.com/learn/openid-connect#optional-endpoints) |

[***8.1 Authorization endpoint***](http://connect2id.com/learn/openid-connect#authz-endpoint)

This is the OP server endpoint where the user is asked to [authenticate](http://connect2id.com/learn/openid-connect#auth-methods) and grant the client app access to the user’s identity (ID token) and potentially other requested details, such as email and name (called UserInfo claims).

This is the only endpoint where the user needs to interact with the OP, via a user agent, which role is typically assumed by the web browser.

[***8.2 Token endpoint***](http://connect2id.com/learn/openid-connect#token-endpoint)

The token endpoint authenticates the client app, then lets it exchange the code received from the [authorisation endpoint](http://connect2id.com/learn/openid-connect" \l "authz-endpoint) for an ID token and access token.

The token endpoint it may also accept other OAuth 2.0 [grant types](http://tools.ietf.org/html/rfc6749#section-1.3) to issue tokens, for example:

* [JWT assertion](http://tools.ietf.org/html/rfc7523)
* [SAML 2.0 assertion](http://tools.ietf.org/html/rfc7522)

[***8.3 UserInfo endpoint***](http://connect2id.com/learn/openid-connect#userinfo-endpoint)

The [UserInfo endpoint](http://openid.net/specs/openid-connect-core-1_0.html" \l "UserInfo) returns previously consented user profile information to the client app. For that a valid access token is required.

GET /userinfo HTTP/1.1

Host: openid.c2id.com

Authorization: Bearer SlAV32hkKG

The UserInfo is JSON encoded and may optionally be packaged as a JWT that is signed or encrypted.

HTTP/1.1 200 OK

Content-Type: application/json

{

"sub" : "alice",

"email" : "alice@wonderland.net",

"email\_verified" : true,

"name" : "Alice Adams",

"picture" : "https://c2id.com/users/alice.jpg"

}

[***8.4 Optional endpoints***](http://connect2id.com/learn/openid-connect#optional-endpoints)

OpenID Connect providers can have these additional endpoints:

* [**WebFinger**](http://openid.net/specs/openid-connect-discovery-1_0.html#IssuerDiscovery) — Enables dynamic discovery of the OpenID Connect provider for a given user, based on their email address or some other information.
* [**Provider metadata**](http://openid.net/specs/openid-connect-discovery-1_0.html#ProviderMetadata) — JSON document listing the OP endpoint URLs and the OpenID Connect / OAuth 2.0 server features that it supports. Client apps can use this information to configure their requests to the OP.
* **Provider JWK set** — JSON document containing the provider’s public keys (typically RSA) in JSON Web Key (JWK) format. These keys are used to secure the issued ID tokens and other artifacts.
* [**Client registration**](http://openid.net/specs/openid-connect-registration-1_0.html) — RESTful web API for registering client apps with the OP. Registration may be protected (require pre-authorisation) or open (public).
* [**Session management**](http://openid.net/specs/openid-connect-session-1_0.html) — Enables client apps to check if a logged in user has still an active session with the OpenID Connect provider. Also to facilitate logout.

[**9. Where is the spec?**](http://connect2id.com/learn/openid-connect#specs)

The standard was produced by a working group at the [OpenID foundation](http://openid.net/) and comprises a set of documents which can be explored at <http://openid.net/connect/>.

[**10. FAQ**](http://connect2id.com/learn/openid-connect#faq)

[***10.1 How is OpenID Connect related to OAuth 2.0?***](http://connect2id.com/learn/openid-connect#oidc-oauth-relationship)

OAuth 2.0 is an authorization protocol for obtaining access tokens for web APIs and protected resources. OpenID Connect utilizes the OAuth 2.0 semantics and flows to allow client apps to access the user’s identity, encoded in a JSON Web Token (JWT) called ID token.

[***10.2 How do OpenID providers authenticate users?***](http://connect2id.com/learn/openid-connect#auth-methods)

OpenID Connect leaves this entirely up to the particular IdP. Implementers may use any method for authenticating users, or combine two methods for stronger security (2FA).

* Username / password
* Hardware tokens
* SMS confirmation
* Biometrics
* Etc.

The IdP may set the optional acr and amr claims in the [ID token](http://connect2id.com/learn/openid-connect#id-token) to inform the client how the user has been authenticated.

Clients are also have control over authentication:

* The optional prompt=login parameter will cause the user to be (re)authenticated, even if they have a valid session (cookie) with the IdP.
* With IdPs that support various authentication strengths, the application may request stronger authentication using the optional acr\_values parameter.

[***10.3 What is a bearer access token?***](http://connect2id.com/learn/openid-connect#bearer-access-token)

The access token resembles the concept of a physical token or ticket. It permits the carrier access to a specific HTTP resource or web service, which is typically limited by scope and has an expiration time.

Bearer access tokens are easy to use - whoever has them has a key to the protected resource. For that reason they must always be passed around over a secure channel (TLS / HTTPS) and stored securely.

OpenID Connect employs OAuth 2.0 access tokens to allow client apps to retrieve consented [user information](http://connect2id.com/learn/openid-connect#claims) from the [UserInfo endpoint](http://connect2id.com/learn/openid-connect" \l "userinfo-endpoint).

An OpenID provider may extend the access token scope to other protected resources and web APIs.