

OIDC authentication flows

OIDC primer - a course on OpenID Connect

Credits: Roland Hedberg, Ioannis Kakavas



Introduction to OpenID Connect

OIDC defines an interoperable way to perform **user authentication**.

- 1. Clients can **verify the identity** of the end-user based on the authentication performed by an Authorization Server;
- 2. It allows clients to **obtain basic profile information** about the end-user in an interoperable and REST-like manner.

Authentication flows

OpenID supports three flows to authenticate a user and retrieve ID token:

- 1. **Authorisation code flow** the most commonly used flow, intended for traditional web apps as well as native/mobile apps. This flow offers optimal security, as tokens are not revealed to the browser and the client app can also be authenticated.
- 2. **Implicit flow** 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.
- 3. **Hybrid flow** 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 (not shown in the following).

OpenID endpoints

The endpoints defined in the standard are:

- **Authorize endpoint**: this endpoint performs authentication and authorisation.
- **Token endpoint**: this endpoint allows the requester to get his tokens. If the authorize endpoint is human interaction, this endpoint is machine to machine interaction.
- UserInfo endpoint: this endpoint allows you to make a request using your access token to receive claims about the authenticated end-user

Optional endpoints are:

- Discovery: this endpoint provide metadata about the OpenID Connect provider, allowing applications to automatically configure for that provider.
- **Client Registration**: this endpoint allow a relaying party to register with the OpenID provider.

OpenID authentication summary

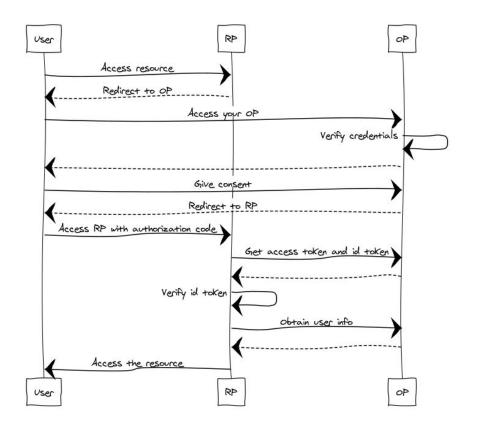
In order to use the OpenID connect authentication:

- register a Client (this can happen dynamically or statically) and obtain a client_id & client_secret
- issue an authentication request to the OP endpoint by redirecting the user browser
- the OP will authenticate user (via username/password or any other mechanism)
- the OP will then redirect the user browser to the Client redirection endpoint and extract the access token
- [request an ID token to the Token Endpoint of the OP using the access token]
- parse the response and extract the id token
- (optionally) use the access token to retrieve user information

Client Registration - Python Example

```
from oic.oic import Client as OIDCClient
from oic.oic.message import AuthorizationResponse, IdToken, Message
from oic.utils.authn.client import CLIENT AUTHN METHOD
def init (self):
    self.flow = 'code'
    self.client = OIDCClient(client authn method=CLIENT AUTHN METHOD)
   # Get the provider configuration information
   provider info = self.client.provider config(self.ISSUER)
    # Register with the provider
   reg endpoint = provider info["registration endpoint"]
    self.client.redirect uris = ["http://localhost:8090/code flow callback", "http://localhost:8090/implicit flow callback"]
    self.client.response types = ["code", "token id token"]
    registration response = self.client.register(reg endpoint)
    # Check registration response
   reg resp = Message()
   reg resp.from dict(dictionary=registration response)
    reg resp.verifv()
```

Authorization code flow



- 1. User access resource on RP.
- The RP redirect the user to the OP for authentication.
- Client sends an Authentication Request containing the desired request parameters to the OP.
- OP Server Authenticates the End-User by checking credentials.
- 5. Authorization Server obtains End-User Consent/Authorization.
- 6. Authorization Server sends the End-User back to the Client with an Authorization Code.
- 7. Client requests a response using the Authorization Code at the Token Endpoint.
- 8. Client receives a response that contains an ID Token and Access Token in the response body.
- 9. Client validates the ID token and retrieves the End-User's Subject Identifier.

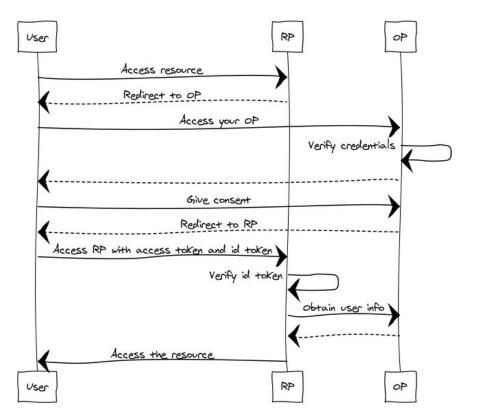
Authorization code flow - Python

```
def authenticate (self, session):
    # Use the session object to store state between requests
    session["state"] = rndstr()
    session["nonce"] = rndstr()
    # Make authentication request
    request args = {
        "client id": self.client.client id,
        "response type": "code",
        "scope": ["openid"],
        "nonce": session["nonce"],
        "redirect uri": self.client.redirect uris[0], # http://localhost:8090/code flow callback
        "state": session["state"]
    auth req = self.client.construct AuthorizationRequest (request args=request args)
    login url = auth req.request (self.client.authorization endpoint)
    return login url
```

Authorization code flow - Python, callback

```
def code flow callback(self, auth response, session):
    # Parse the authentication response
    aresp = self.client.parse response(AuthorizationResponse, info=auth response, sformat="urlencoded")
    assert aresp["state"] == session["state"]
    # Make token request
    access code = aresp["code"]
    args = {
        "code": access code,
        "redirect uri": self. get redirect uris for auth(),
        "client id": self.client.client id,
        "client secret": self.client.client secret
    resp = self.client.do access token request(scope=aresp["scope"],
                                               state=aresp["state"],
                                               request args=args,
                                               authn method="client secret post")
    # Validate the ID Token according to the OpenID Connect spec (sec 3.1.3.7.)
    id token claims = IdToken()
    id token claims.from dict(dictionary=resp['id token'])
    id token claims.verify()
    # Make userinfo request
    userinfo = self.client.do_user_info_request(state=aresp["state"])
    # Set the appropriate values
    access token = resp['access token']
    return success page (access code, access token, id token claims, userinfo)
```

Implicit flow



- Client prepares an Authentication Request containing the desired request parameters.
- 2. Client sends the request to the Authorization Server.
- 3. Authorization Server Authenticates the End-User.
- Authorization Server obtains End-User Consent/Authorization.
- 5. Authorization Server sends the End-User back to the Client with an ID Token and, if requested, an Access Token.
- 6. Client validates the ID token and retrieves the End-User's Subject Identifier.

Implicit flow - Python

```
def authenticate (self, session):
    # Use the session object to store state between requests
    session["state"] = rndstr()
    session["nonce"] = rndstr()
    # Make authentication request
    request args = {
        "client id": self.client.client_id,
        "response type": ["id token", "token"],
        "scope": ["openid"],
        "nonce": session["nonce"],
        "redirect uri": self.client.redirect uris[1], # http://localhost:8090/implicit flow callback
        "state": session["state"]
    1
    auth req = self.client.construct AuthorizationRequest (request args=request args)
    login url = auth req.request(self.client.authorization endpoint)
    return login url
```

Implicit flow - Python, callback

```
def implicit flow callback (self, auth response, session):
    # Parse the authentication response
    aresp = self.client.parse response(AuthorizationResponse, info=auth response, sformat="urlencoded")
    assert aresp["state"] == session["state"]
    # Validate the ID Token according to the OpenID Connect spec (sec 3.2.2.11.)
   id token claims = IdToken()
   id token claims.from dict(dictionary=aresp['id token'])
   id token claims.verify()
    # Make userinfo request
   userinfo = self.client.do user info request(state=aresp["state"])
    # Set the appropriate values
    access code = None
    access token = aresp['access token']
   return success page (access code, access token, id token claims, userinfo)
```

DETA

Thanks for your attention!