Course by Consortium GARR and Reti Business & IT Consulting

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

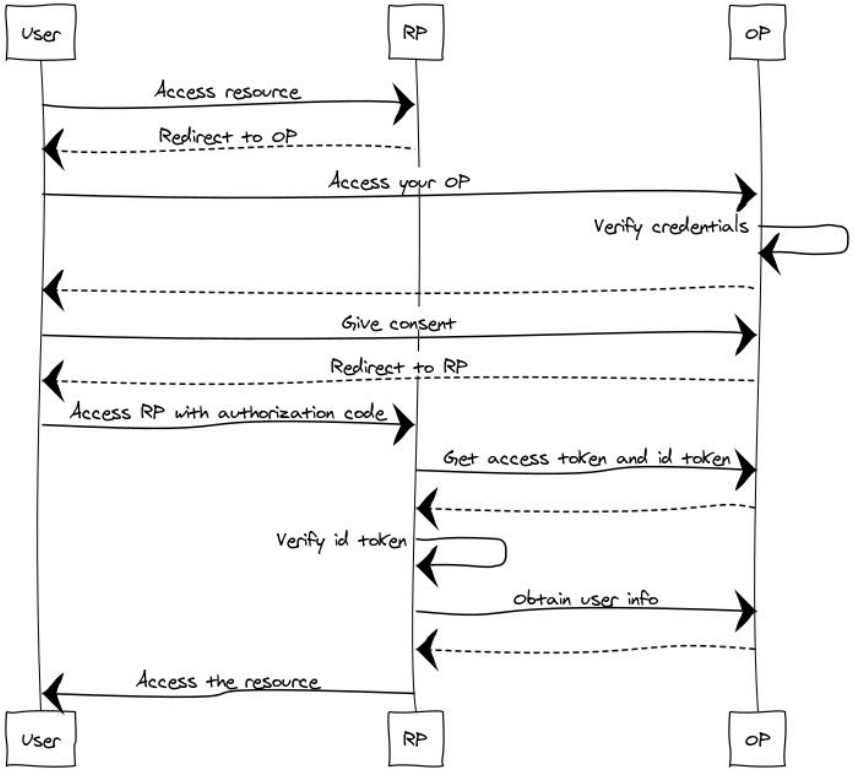
In order to use the OpenID connect authentication:

- 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.verify()
```

Authorization code flow



1. User access resource on RP.
2. The RP redirect the user to the OP for authentication.
3. Client sends an Authentication Request containing the desired request parameters to the OP.
4. 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.

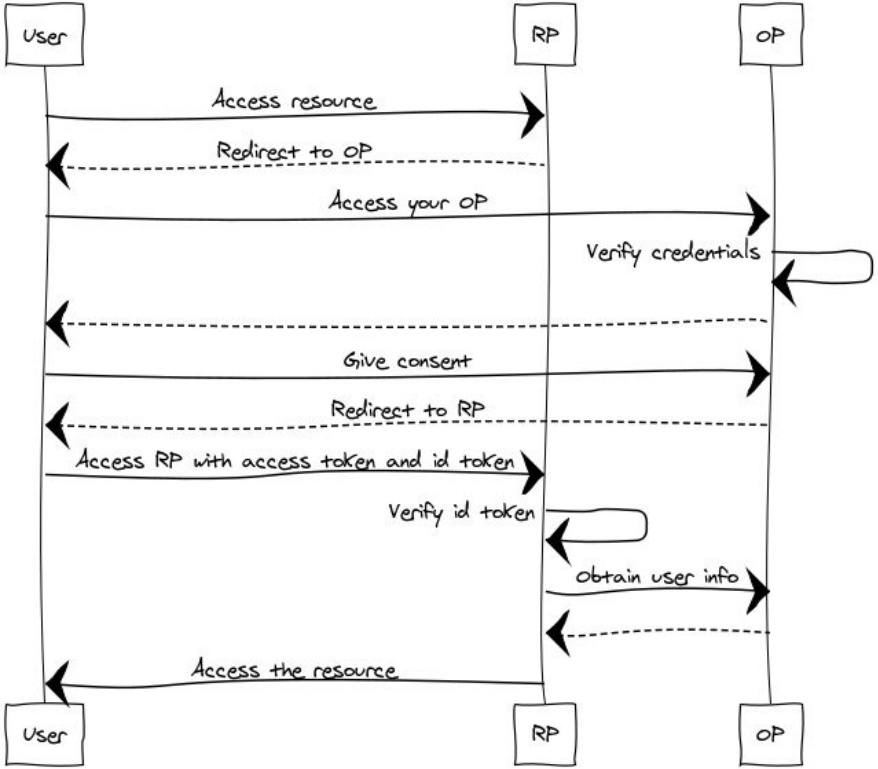
Authorization code flow - Python

```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

```python
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



1. 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.
4. 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

```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"]
    }

    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

```python
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)
```

Q&A

Thanks for your attention!