The OpenID Connect Protocol
GET /summary

{
    "part1":"Some words on OAuth 2.0",
    "part2":"The OpenID Connect Protocol",
    "part3":"OpenID Connect VS SAML",
    "part4":"Support of OpenID Connect in LL::NG"
}
RFC 6749

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.
Roles

- Resource owner (end-user)
- Client (third-party)
- Authorization Server
- Resource Server
Authorization Grant

**Authorization Code**
- More secure
- Server side applications
- Tokens hidden to end user

**Implicit**
- Access token directly sent
- Designed for JS client application

**Resource Owner Password Credentials**
- Requires high trust between end-user and client

**Client credentials**
- Client is often the resource owner
Tokens

Access Token:
- Opaque
- Limited duration
- Scope
- Give access to the resource server

Refresh Token:
- Allow to get a new access token
- Optional
- Can not be used as an access token
Client Registration

- Client has to be registered with the authorization server
- OAuth 2.0 do not specify how this registration is done
- Information that should be registered:
  - Client type
  - Redirection URIs
  - Other: application name, logo, etc.
- The client then received a client_id and a client_password
Client types

- **Confidential**: Clients capable of maintaining the confidentiality of their credentials:
  - Application on a secure server

- **Public**: Clients incapable of maintaining the confidentiality of their credentials:
  - Native mobile application
  - Web browser based application
Endpoints

• Authorization Server:
  – Authorization: where the resource owner gives authorization
  – Token: where the client get tokens

• Client:
  – Redirection: where the resource owner is redirected after authorization
Authorization

GET /authorize?
response_type=code&client_id=s6BhdRkqt3&state=xyz&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

https://client.example.com/cb?
code=SpIXlOBeZQQYbYS6WxSbIA&state=xyz
Token

POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic
czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSblA&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcbl
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "example",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOKfOxG5Qx2TIKWIA",
  "example_parameter": "example_value"
}
Resource

GET /resource/1 HTTP/1.1
Host: example.com
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
(1) AuthN Request
(2) AuthN & AuthZ
(3) AuthN Response
(4) UserInfo Request
(5) UserInfo Response
Built on top of OAuth 2.0

- **Flows:**
  - Based on OAuth 2.0 Authorization grants:
    - Authorization Code
    - Implicit
  - New flow: Hybrid

- **Scope:**
  - New scope: “openid”

- **Endpoints:**
  - Use Authorize, Token and Redirection endpoints
  - New endpoint: UserInfo

- **Tokens:**
  - Use access and refresh tokens
  - New token: ID token (JWT)
OpenID Connect Protocol Suite

- Core
- Discovery
- Dynamic Client Registration
- Session Management
- Form Post Response Mode

- Minimal
- Dynamic
- Complete
Underpinnings

OAuth 2.0 Core
OAuth 2.0 Bearer
OAuth 2.0 Assertions
OAuth 2.0 JWT Profile
OAuth 2.0 Responses

JWT
JWS
JWE
JWK
JWA
WebFinger

JOSE
JOSE

Javascript Object Signing and Encryption
JWT

- Concatenation with dots of:
  - base64(Header)
  - base64(Payload)
  - base64(Signature)
eyJhbGciOjI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkF1c2VyIiwibmFtZT0iSGFzdCJ9.eyJuYXZlIjoiMjAxIiwic2NvcGUiOjEwMCwiaWF0Y29uIjoiMjAxMC0xMBoiLCJleHAiOjEwMjQ4ODE2MDQsImlzcyI6Inp5cmNlIiwiYXV0b24iOjB9

http://jwt.io/

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

```javascript
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)
```
http://auth.example.com/oauth2/authorize?
response_type=code
&client_id=lemonIdap
&scope=openid%20profile%20email
&redirect_uri=http%3A%2F%2Fauth.example.com%2Foauth2.pl
%3Fopenidconnectcallback%3D1
&state=ABCDEFGHIJKLMNOPQRSTUVWXXZ
Authentication required

- LDAP
- SAML
- OpenID Connect
- CAS

Login:
coudot

Password:
****

Check my last logins

Connect

Reset my password
Create an account

Service provided by LemonLDAP:NG free software covered by the GPL license.
The application Sample would like to know:

- Your identity
- Your profile
- Your email

Service provided by LemonLDAP::NG free software covered by the GPL license.
http://auth.example.com/oauth2.pl?
openidconnectcallback=1;
code=f6267efe92d0fc39bf2761c29de44286;
state=ABCDEFGHIJKLMNOPQRSTUVWXYZ
POST /oauth2/token HTTP/1.1
Host: auth.example.com
Authorization: Basic xxxx
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&code=f6267efe92d0fc39bf2761c29de44286
&redirect_uri=http%3A%2F%2Fauth.example.com%2Foauth2.pl%3Fopenidconnectcallback%3D1
{"id_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJhY3lOiJsb2EtM1lsImF1dGhfdGlzZSI6MTQzMjExMzU5Mjw0IiwidXNlcl9pZCI6MywiZXhwIjoxNTMxMzU5MzY5fQ==", "expires_in": "3600", "access_token": "512cdb7b97e073d0656ac9684cc715fe", "token_type": "Bearer"}
ID Token payload

```json
{
    "acr": "loa-2",
    "auth_time": 1432113593,
    "iat": 1432113966,
    "at_hash": "9axzsNi9pNDk5zWefKsM6A",
    "iss": "http://auth.example.com/",
    "exp": "3600",
    "azp": "lemonldap",
    "nonce": "1234567890",
    "sub": "coudot@linagora.com",
    "aud": ["lemonldap"
    ]
}
```
POST /oauth2/userinfo HTTP/1.1
Host: auth.example.com
Authorization: Bearer 512cdb7b97e073d0656ac9684cc715fe
Content-Type: application/x-www-form-urlencoded
{  "name": "Clément OUDOT",  "email": "coudot@linagora.com",  "sub": "coudot@linagora.com"}
OpenID Connect vs SAML
Frameworks

- REST
- JSON
- JWT/JOSE
- HTTP GET/POST
- Offline mode possible

- SOAP
- XML
- XMLSec
- HTTP GET/POST
- No offline mode
Network flows

- Direct connection between RP and OP required
- Request can be passed as reference (Request URI)
- Always RP initiated

- Can work without link between SP and IDP
- Request and responses can be passed as references (Artefacts)
- IDP initiated possibility
Configuration

- Published as JSON (openid-configuration)
- Client (RP) registration needed
- Keys publication (jwks)

- Published as XML (metadata)
- SP and IDP registration needed
- Keys publication (metadata)
Security

- HTTPS
- Signature and encryption of JWT

- HTTPS
- Signature and encryption of all messages
User consent

- Consent required to authorize requested scopes
- No account federation

- No consent needed to share attributes
- Consent can be asked to federate accounts
Implementation

- RP: quite easy
- OP: difficult
- SP: difficult
- IDP: difficult
LemonLDAP::NG

- Free Software (GPLv2+) / OW2 consortium
- Single Sign On, Access Control
- Service Provider / Identity Provider
- Perl/Apache/CGI/FCGI
- Lost Password and Account Register self services
- [http://www.lemonldap-ng.org](http://www.lemonldap-ng.org)
OpenID Connect RP

• Authorization Code Flow
• OP selection screen
• JSON configuration and JWKS parsing
• Full configuration of authentication requests (scope, display, prompt, acr_values, etc.)
• Attributes mapping
OpenID Connect OP

- Authorization Code / Implicit / Hybrid Flows
- Signature: HS256, HS384, HS512, RS256, RS384, RS512
- Token endpoint authentication
- JSON configuration and JWKS publication
- Configuration of Authentication Contexts
- Attributes mapping
Seems all clear

Any question?